PROJECTE FINAL DE CARRERA

IMPLEMENTACIÓ ORIENTADA A LA DOCÈNCIA DE PROTOCOLS DE COMUNICACIÓ

(TEACHING-ORIENTED IMPLEMENTATION OF COMMUNICATION PROTOCOLS)

Estudis: Enginyeria de Telecomunicació

Autor: Ramon Gallardo Burguet

Director/a: Marcel Fernández Muñoz

Any: 2016

Índex General

Col·laboracions

Agraïments

Dedico aquest projecte al meu fill Oriol i a la meva dona Eva que són el que estimo més en aquest món i amb el seu suport en tot moment m'han ajudat a seguir endavant. El seu somriure és el motor del meu cor.

Agraeixo a tota la meva família per estar sempre al meu costat i donar-me suport en tots els moments difícils. Gràcies pares i germans per la vostra ajuda.

Finalment també vull donar les gràcies a en Marcel Fernández pel seu suport i comprensió alhora de fer aquest projecte i també a les seves classes que em van fer interessar en la programació i voler emprendre aquest projecte.

Resum del Projecte

El present projecte està enfocat a desenvolupar un simulador de comunicacions per xarxa basat amb el model d'interconnexió o torre de protocols.

Aquesta eina està orientada a la docència, ja que disposarem d'un entorn didàctic on podrem realitzar la nostra pròpia confecció de la xarxa i llavors podrem observar com es realitza aquesta comunicació entre els diferents elements de la nostra simulació, a través de les diverses capes o nivells que posseeix la comunicació.

Aquest entorn de simulació s'ha dissenyat de manera que sigui molt visual, simple i amb moltes funcionalitats. Per tant, els usuaris no tindran cap dificultat alhora d'utilitzar aquesta eina i al ser un entorn didàctic podran aprendre i posar en pràctica diferents situacions de xarxes de comunicacions.

El nostre simulador està dissenyat en un entorn web on utilitzem llibreries molt gràfiques amb tecnologies intuïtives que faciliten la usabilitat de l'usuari. Al ser una plataforma web tenim un entorn on varis usuaris poden accedir al mateix moment i crear cada un la seva pròpia confecció personalitzada. Això fa que cada usuari que usa el simulador té una connexió independent amb el servidor central del nostre projecte.

És una plataforma client-servidor on la comunicació entre ells es realitza mitjançant una nova llibreria de HTML5, els Websockets. A partir d'aquesta llibreria podem crear una comunicació bidireccional on client i servidor s'intercanvien informació sobre la simulació creada. Com ja hem comentat en l'entorn web del client es crea la confecció i en el servidor s'encarrega de simular aquesta xarxa de comunicació, respectant el model d'interconnexió.

La torre de protocols és un model d'estandardització que es divideix en capes o nivells i cada un d'aquests s'ocupa de realitzar una tasca en concret per poder realitzar correctament la comunicació entre els diferents elements de la xarxa.

Per tant, en el servidor, a través d'una plataforma Java es crearan tots els elements necessaris per poder realitzar la simulació i s'enviarà els diferents comportaments de cada protocol emprat de cada nivell de la comunicació a l'entorn web del client, per a què l'usuari pugui visualitzar l'evolució de la seva simulació.

També cal ressaltar que el servidor utilitza una base de dades per unes funcions concretes. Els usuaris que utilitzen el simulador es poden registrar en la plataforma i tenir un usuari i contrasenya. A partir d'aquí, un cop registrat, l'usuari pot emmagatzemar les seves simulacions a la base de dades per a un ús a posteriori.

Per tant, el present projecte combina diferents plataformes de programació per arribar a crear una eina multiusuari didàctica destinada als protocols de comunicació.

Resumen del Proyecto

El presente proyecto está enfocado a desarrollar un simulador de comunicaciones de red basado en el modelo de interconexión o torre de protocolos.

Esta herramienta está orientada a la docencia, ya que dispondremos de un entorno didáctico dónde podremos realizar nuestra propia confección de red y luego observaremos como se realiza esta comunicación entre los diferentes elementos de nuestra simulación, a través de las múltiples capas o niveles que posee la comunicación.

Este entorno de simulación se ha diseñado de manera que sea muy visual, simple y con muchas funcionalidades. Por lo tanto, los usuarios no van a tener ninguna dificultad para utilizar esta herramienta y al ser un entorno didáctico podrán aprender y poner en práctica diferentes situaciones de redes de comunicaciones.

Nuestro simulador está diseñado en un entorno web donde utilizaremos librerías muy gráficas con tecnologías intuitivas que facilitan la usabilidad del usuario. Al ser una plataforma web tenemos un entorno donde varios usuarios pueden acceder a la vez y crear cada uno su propia confección personalizada. Esto hace que cada usuario que usa el simulador tiene una conexión independiente con el servidor central del proyecto.

Es una plataforma cliente-servidor donde la comunicación entre ellos se realiza mediante una nueva librería de HTML5, los Websockets. A partir de esta librería podemos crear una comunicación bidireccional donde cliente y servidor se intercambian información sobre la simulación creada. Como ja hemos comentado en el entorno web del cliente, se crea la confección y en el servidor, se encarga de simular esta red de comunicación, respetando el modelo de interconexión.

La torre de protocolos es un modelo de estandarización que se divide en capas o niveles y cada uno de estos se ocupa de realizar una tarea concreta para

poder realizar correctamente la comunicación entre los diferentes elementos de la red.

Por lo tanto, en el servidor, a través de una plataforma Java se crearán todos los elementos necesarios para poder realizar la simulación y se enviará los diferentes comportamientos de cada protocolo utilizado en cada nivel de la comunicación al entorno web del cliente, para que el usuario puede visualizar la evolución de su simulación.

También cabe resaltar que el servidor utiliza una base de datos para unas funciones concretas. Los usuarios que utilizan el simulador se pueden registrar en la plataforma y tener un usuario y contraseña. A partir de aquí, una vez registrado, el usuario puede guardar sus simulaciones en la base de datos para un uso a posteriori.

Por lo tanto, el presente proyecto combina diferentes plataformas de programación para llegar a crear una herramienta multiusuario didáctica destinada a los protocolos de comunicación.

Abstract

This project is aimed at developing a Communications network Simulator based on the interconnection model or protocol stack.

This tool is teaching-oriented application, because we will have a learning environment where we can make our own manufacturing network. Then we can observe how this communication between the different elements of our simulation is performed through multiple layers that has the communication.

This simulation environment is designed so that it is very visual, simple and with many features. Therefore, users will not have any difficulty using this tool and being a learning environment can study and implement different situations of communications networks.

Our simulator is designed in a web environment where we use very graphic libraries with intuitive technologies that facilitate the usability of the user. Being a web platform we have an environment where multiple users can simultaneously access and each create his own custom network. This makes each user that use the simulator has a separate connection to the central server of the project.

It is a client-server platform where communication between them is done through a new library of HTML5, the Websockets. From this library we can create a two-way communication where client and server exchange information of the simulation created. As we discussed, in the web client environment, the network is created and then the server is responsible for simulating this communication network, respecting the networking model.

Tower protocols is a standardization model that is divided in layers and each of it deals with a specific task to correctly perform communication between different network elements.

Therefore, on the server, through a Java platform all the necessary elements will be created to perform the simulation and will send the different

behaviours of each protocol each level of communication to the client's web environment, so the user can view the evolution of its simulation.

It should also be noted that the server uses a database for particular functions. Users who use the simulator can be registered on the platform and have a username and password. From here, once registered, the user can save their simulations in the database for subsequent use.

Therefore, this project combines different programming platforms to reach multiuser create a teaching tool for communication protocols.

1. Introducció

El present projecte de final de carrera pretén desenvolupar un simulador de comunicacions per xarxa basat amb el model d'interconnexió o torre de protocols. Amb aquest simulador podrem realitzar la nostra pròpia confecció de la xarxa i llavors podrem observar com es realitza aquesta comunicació entre els diferents elements de la nostra simulació, a través de les diverses capes o nivells de la torre de protocols.

1.1. Context del projecte

Fa anys que em dedico professionalment en un entorn on desenvolupem pàgines web i també donem suport a tots els problemes en les xarxes de comunicacions.

Mitjançant aquest projecte pretenc posar en pràctica aquests dos conceptes i confeccionar una solució web on l'element principal siguin les xarxes de comunicació.

1.2. Objectius

El projecte té com a objectiu principal crear una eina amb les noves tecnologies que sigui didàctica i que es pugui simular diferents situacions amb xarxes de comunicació. Per tant, crear un simulador de protocols de comunicació.

A més a més, aquest projecte té com a objectiu utilitzar noves tecnologies com els Websockets. Ja que és una eina potent que ens permetrà realitzar la comunicació entre client-servidor i és una llibreria nativa de HTML5 que tots els navegadors actuals posseeixen.

Com a resultat volem aconseguir un simulador de protocols de comunicació que sigui visual i que es pugui utilitzar des de qualsevol navegador web. A més, poder fer que aquesta plataforma sigui multiusuari i que vàries persones puguin utilitzar l'eina al mateix moment.

1.3. Estructura de la memòria

En la present memòria explicaré les diferents plataformes de programació que he utilitzat per a realitzar el projecte. Classificarem la informació a partir del codi emprat en l'entorn del client o usuari que utilitza l'entorn gràfic, i el codi emprat en el servidor. També explicaré les tecnologies emprades per a cada plataforma i per a què ens serveixen.

Al final acabaré parlant de les possibles línies futures que es podrien realitzar a partir del meu projecte o possibles modificacions que ara faria al projecte un cop acabat.

Com a punt i final per entendre millor el comportament d'aquesta aplicació adjuntarem com a annexa una simulació d'exemple.

2.Tecnologies emprades

Com ja hem vist amb anterioritat l'aplicació és una plataforma client-servidor i per tant tenim dues parts diferenciades: l'entorn web del client i l'entorn en Java del servidor. Les dues parts corren sobre un servidor web implementat amb Apache Tomcat, que inclou un servidor HTTP i la màquina virtual de Java.

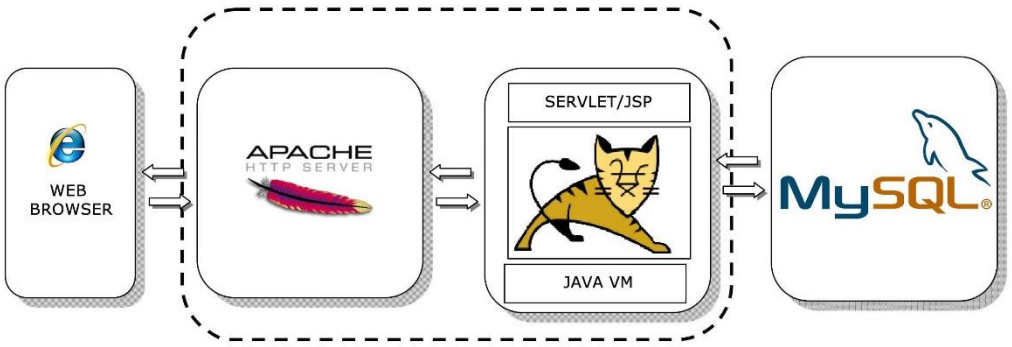

Fig.2.1 – Estructura client-servidor

Apache Tomcat és un contenidor de Servlets desenvolupat per "Apache Software Foundation". Un Servlet és un objecte Java que posseeix unes propietats i mètodes que interaccionen a les peticions HTTP. D'aquesta manera, els objectes Servlet són capaços de rebre una invocació i generar una resposta en funció de les dades rebudes. Per tant, tenim que a partir de peticions HTTP obtenim execucions amb Java en el servidor o contenidor de Servlets. (2) (3) (7)

A més a més en el nostre cas el servidor utilitza una base de dades per emmagatzemar informació del usuaris i les seves configuracions. La base de dades utilitzada és MySQL. Aquest sistema de gestió de bases de dades relacional, utilitza el llenguatge estàndard SQL (Structured Query Language). (9)

Al ser un servidor HTTP tenim que els objectes Servlets del nostre contenidor, responen a les invocacions amb pàgines HTML. Per tant, una part important del projecte correspon a la construcció de les pàgines HTML, on s'utilitza el llenguatge HTML5, la revisió del llenguatge bàsic de la World Wide Web més recent. (15)

Aquesta última entrega HTML ens permetrà utilitzar una API de comunicació bidireccional molt important en el projecte, el Websocket. HTML5 Websocket és una API que ens permetrà realitzar una comunicació entre client i servidor per mantenir la informació actualitzada en tot moment. Per tant, en el servidor també haurem d'utilitzar una llibrera que utilitzarem en la màquina virtual de Java. (11) (16)

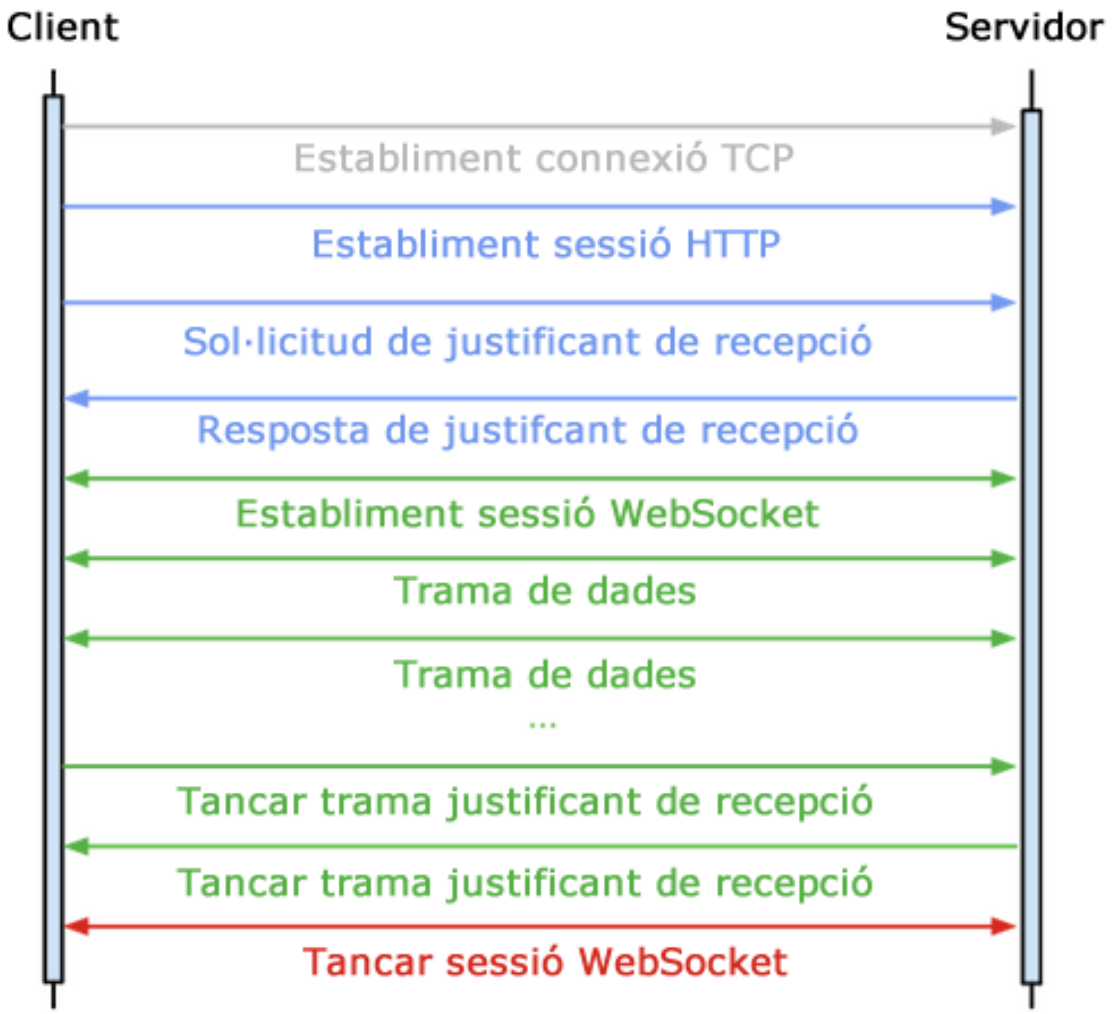

Fig.2.2 – Estructura comunicació WebSocket

Com es pot observar en la figura 2.2, la comunicació es realitza a través d'un socket que crea una connexió punt a punt per on s'enviaran trames de dades amb la informació de la simulació. Per tant tenim una fase inicial d'obertura de la comunicació i per finalitzar hi haurà una fase de tancament, on en ambdós fases client i servidor es transmetran informació sobre la comunicació.

Les dades que transmetrem a través del Websocket les manipularem a través d'un estàndard de text anomenat JSON. És una estructura de dades on podem anotar diferents objectes constituïts amb atributs i valors. Aquest format és molt utilitzat en intercanvis d'informació entre client/servidor en pàgines web. (23)

Juntament amb el llenguatge HTML, també hem emprat llibreries Javascript que ens permeten interaccionar amb el codi de l'entorn web del client: (18)

- **jsPlumb**: la llibreria més destacada en aquest projecte que ens permet crear de manera visual el nostre simulador. Amb aquesta eina l'usuari pot realitzar la confecció de la simulació, creant els objectes i fent les connexions entre ells. (19)
- **jQuery**: la llibreria base que utilitzen totes les altres llibreries i permet simplificar la manera de manipular els objectes, esdeveniments i animacions del document HTML. (20)
- **jQueryUI**: la llibreria que té una col·lecció d'eines gràfiques com ara: botons, quadres de diàleg, pestanyes, calendaris, etc. (21)
- **i18next**: la llibreria que ens permet tenir la interfície del nostre simulador en diferents idiomes. A partir d'un fitxer per a cada idioma i una referència per paraula clau en el codi, la llibreria substitueix aquestes claus per la seva corresponent traducció que troba en el fitxer de l'idioma seleccionat. (22)

3.Implementació

Per parlar sobre la implementació del projecte, ho dividirem en dues parts: la implementació de l'aplicació i la implementació de les tecnologies. Dins de cada apartat ho classificarem entre l'entorn del client o l'entorn del servidor.

Però abans de començar amb cada una de les parts. Hem d'explicar prèviament que aquest projecte es basa principalment en la implementació de la torre de protocols o model TCP/IP.

El model TCP/IP és un sistema basat en capes o nivells on cada un utilitzarà un protocol establert que realitza un tasca concreta dins la comunicació entre elements.

Els diferents nivells que disposa el model són els següents: Aplicació, Transport, Xarxa, Accés al medi.

A continuació podem veure en la figura 3.1 una connexió entre dos nodes amb els models TP/IP de cada un d'ells.

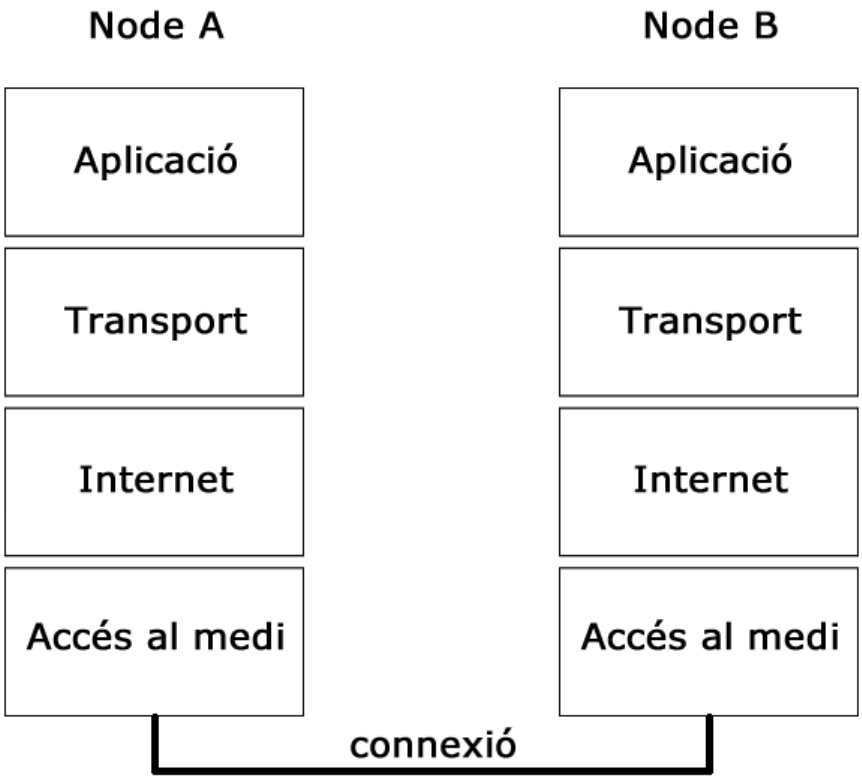

Fig.3.1 – Nivells torre de protocols TCP/IP

Podem observar quin és l'ordre de les capes sent l'Accés al medi la de nivell més baix i la d'Aplicació la de nivell més alt.

El model es realitza a través de capes per a simplificar i agrupar les funcions i procediments que s'han de dur a terme per aconseguir un intercanvi fiable de dades entre dos nodes. Per tant, d'aquesta manera es distribueix les tasques entre les capes tenint en compte que les capes segueixen un ordre jeràrquic, i per tant segueixen un ordre de prioritats.

Aquest model és una arquitectura més simple, però alhora més evolucionat, d'un altre model anomenat OSI. En aquest model predecessor es disposa de set nivells o capes que distribueixen de la següent manera: Aplicació, Presentació, Sessió, Transport, Xarxa, Enllaç i Física. (1)

A continuació podem veure en la figura 3.2 la comparació el model OSI i el model TCP/IP.

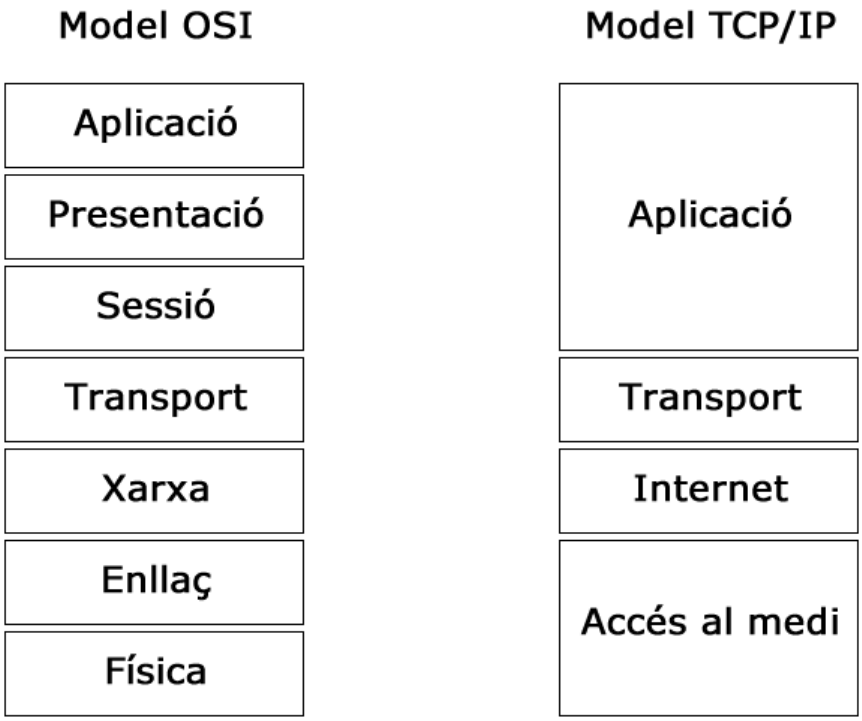

Fig.3.2 – Comparació nivells OSI i TCP/IP

En aquesta comparació podem observar com la capa d'Aplicació del model TCP/IP engloba les tres campes Aplicació, Presentació i Sessió del model OSI, i de la mateixa manera la cap d'Accés al medi del model TCP/IP engloba les

capes Enllaç i Física del model OSI. Aquestes agrupacions són fruit d'una evolució del model TCP/IP respecte el model OSI. (1)

Més endavant veurem com implementem aquest model TCP/IP i quins protocols i nivells utilitzarem.

3.1. Implementació de l'aplicació

L'aplicació del projecte, com ja hem vist, consta d'un simulador HTML que utilitza el client de l'aplicació i d'un servidor web amb màquina virtual de Java que rep les peticions per continuar amb les corresponents simulacions.

3.1.1. Entorn Web del client

En l'entorn web que utilitza l'usuari és on es confecciona la simulació que es vol emprar. A continuació analitzarem aquest espai i les seves funcions.

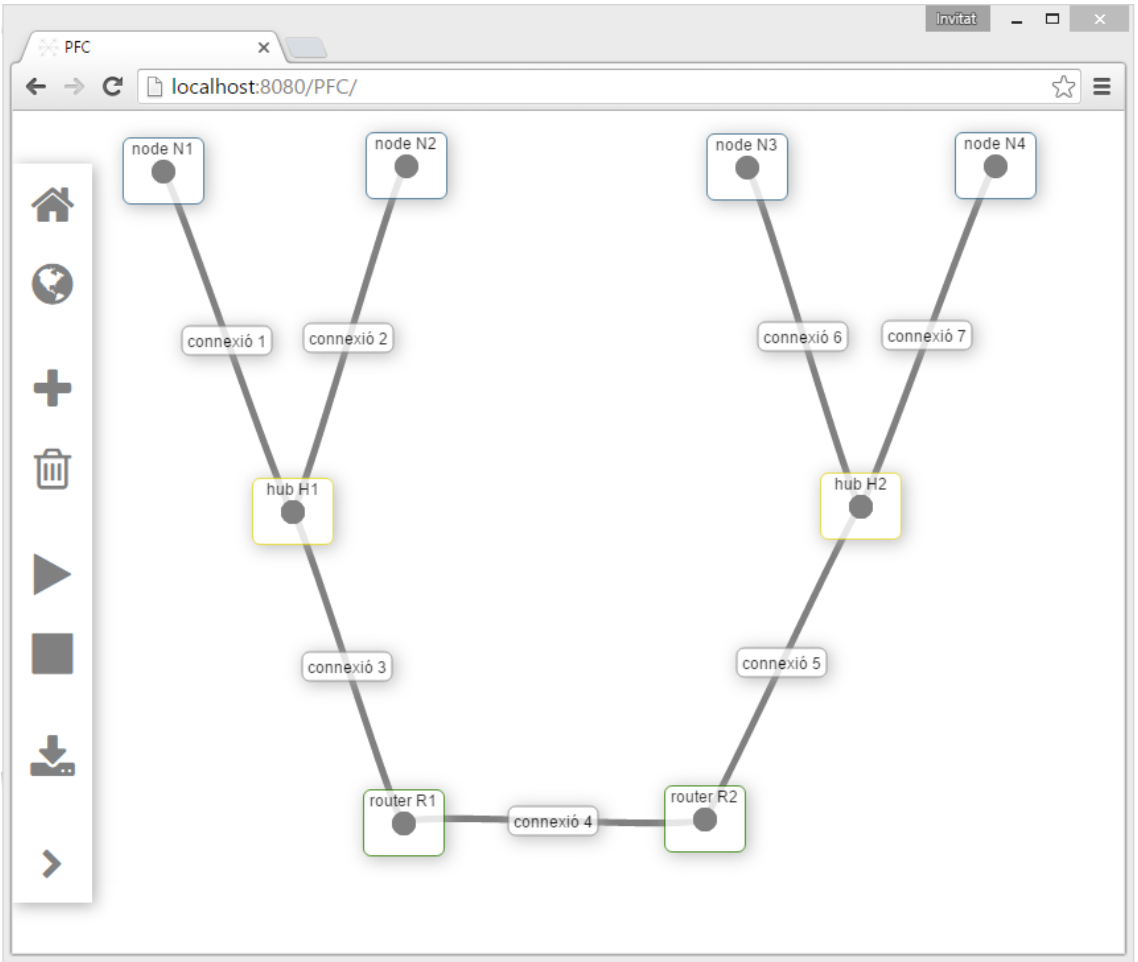

Fig.3.3 – Exemple de simulació amb l'entorn web

En la figura 3.3 veiem un exemple de simulació que es pot realitzar amb l'entorn web. Veiem que la pàgina web consta bàsicament de dues parts diferenciades: la paleta de configuració i l'espai de disseny de la simulació. En la paleta de configuració, que trobarem a la banda esquera de la pantalla, es poden realitzar diferents funcionalitats. Com es pot observar en la figura 3.4 veiem com una de les funcionalitats de la paleta és afegir o esborrar elements i també connexions. Disposarem de tres tipus d'elements: nodes, hubs o routers. Diferenciant-se entre ells pel seu nivell dins el model TCP/IP. Els nodes tenen tots els nivells o capes de la torre TCP/IP, en canvi, els routers tenen fins al nivell de xarxa i els hubs fins al nivell d'enllaç.

A mesura que anem afegint objectes, veurem com aquests van apareixent en l'espai de disseny.

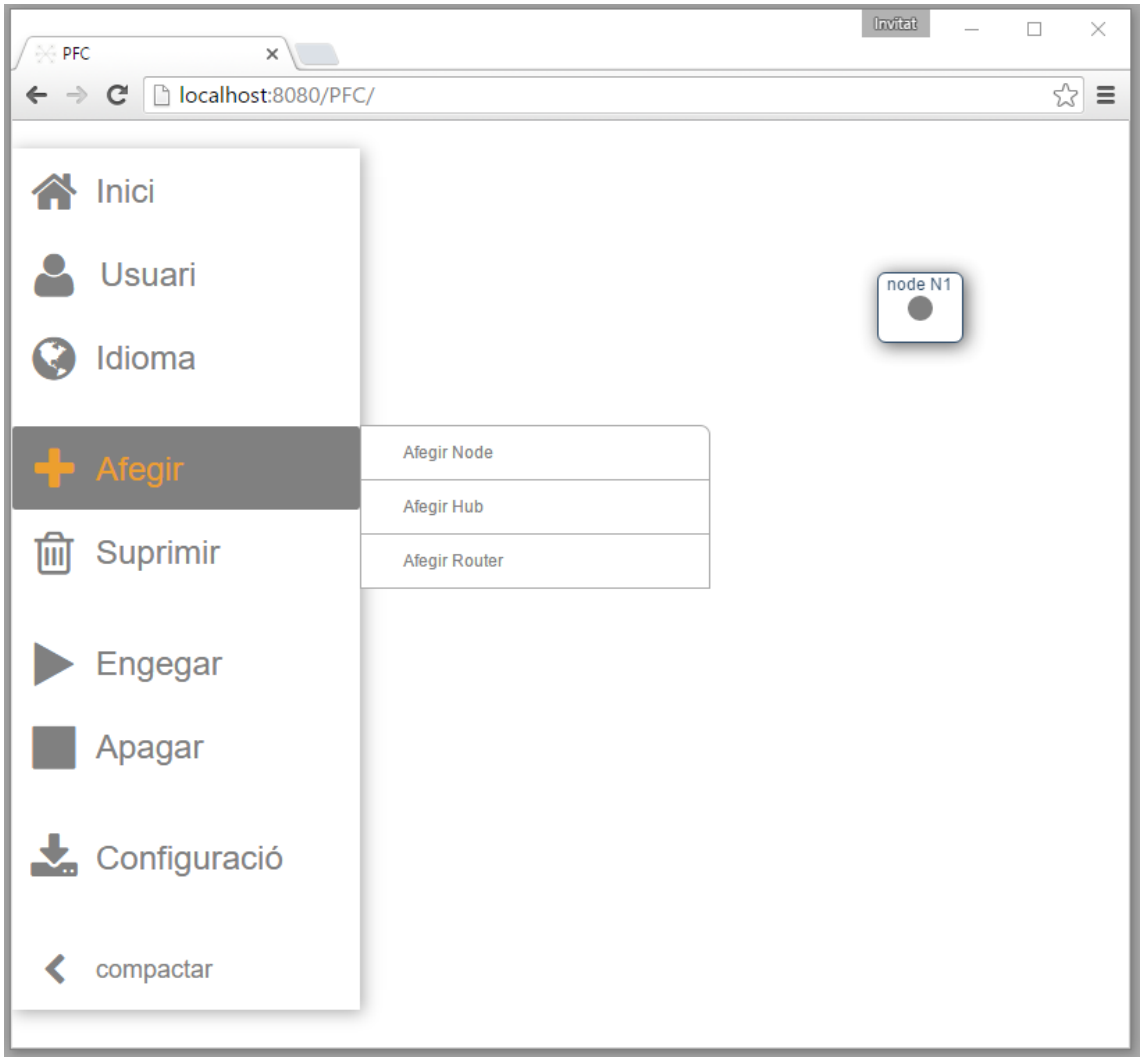

Fig. 3.4 – Entorn web paleta de disseny

Un cop hem afegit diferents elements al nostre disseny, podem crear les connexions o canals de comunicació entre ells mitjançant el concepte "drag and drop" des del centre de cada element, on trobarem un cercle, el cliquem i arrastrem fins a l'element amb el que el vulguem unir. Com podem veure a la figura 3.5 estem creant una connexió de comunicació entre el node N1 amb el hub H1.

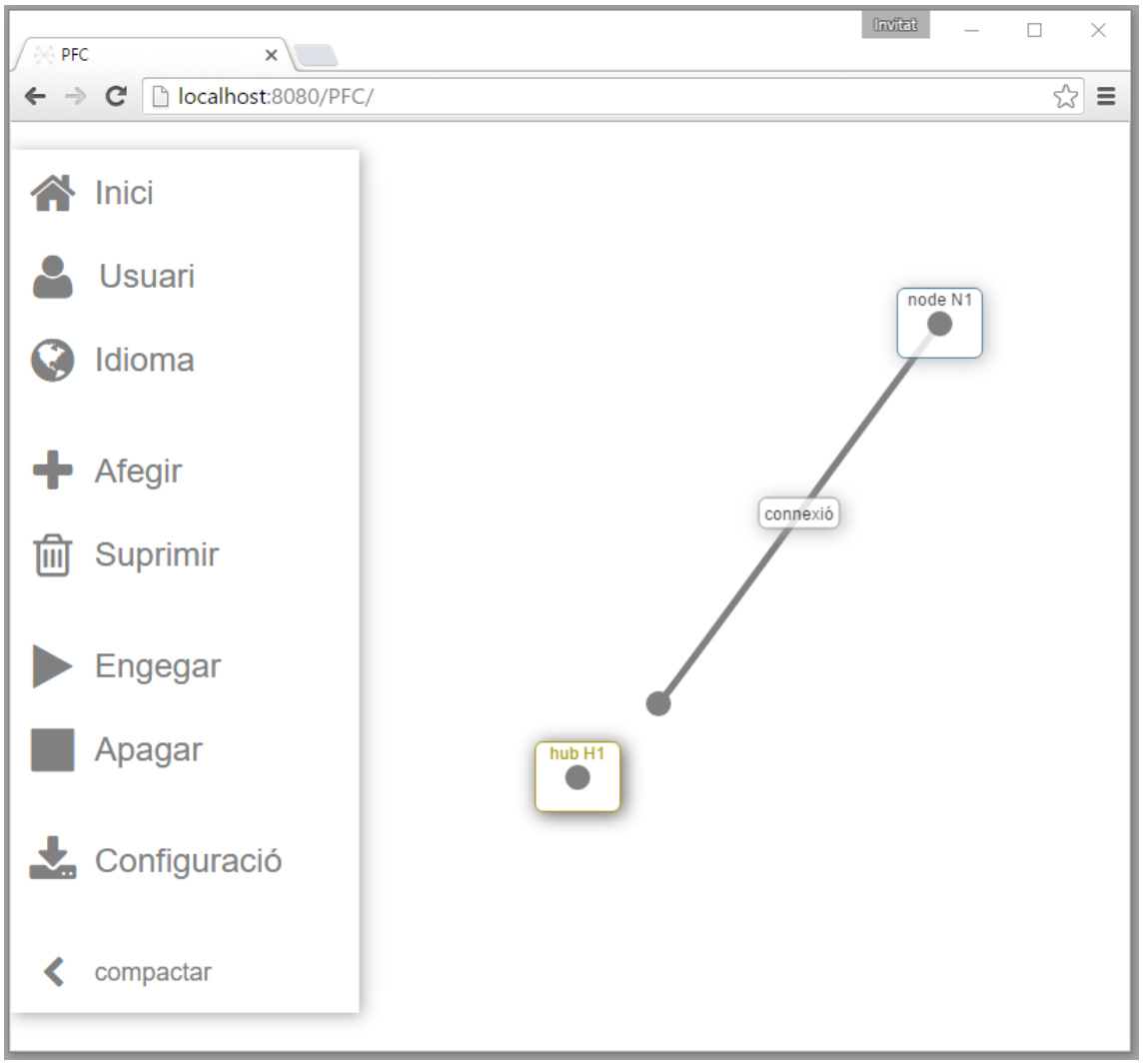

Fig. 3.5 – Entorn web creació d'una connexió

Les connexions entre objectes es poden realitzar sempre i quan compleixin els requisits per a la comunicació. Les possibles connexions que estan permeses són N-H, H-R i R-R, la resta de connexions no són compatibles.

Un cop creat el disseny de la xarxa a analitzar, és hora de marcar quin node serà l'emissor i quin serà el receptor. En aquest cas, el nodes posseeixen un quadre de diàleg propi que s'obre al clicar a sobre de ells. Com es pot veure

en la figura 3.6, marquem el node N1 com a node emissor i el node N2 com a node receptor. A més a més, en el node emissor podem parametritzar quants missatges ha d'enviar i de quina mida en bytes ha de ser cada un.

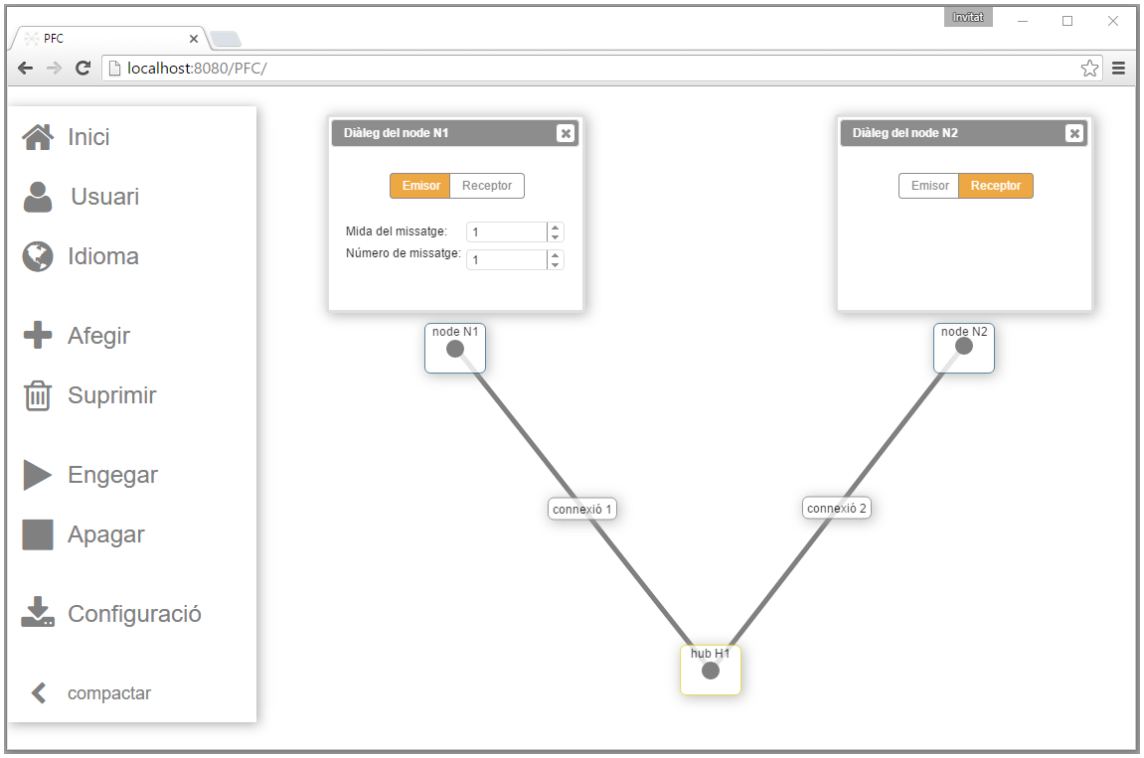

Fig. 3.6 – Entorn web emissor i receptor

En aquest punt, el disseny confeccionat per l'usuari ja està apunt. És a partir, d'aquest moment quan es pot engegar la simulació. Des de la paleta de configuració trobarem aquestes opcions, tant per engegar com per apagar la simulació.

La simulació s'ha d'enviar al servidor i es farà a través de Websockets amb HTML5. Com ja hem parlat en capítols anteriors, es crea una comunicació punt a punt amb el navegador de l'usuari amb el servidor web Apache Tomcat. A través del canal de comunicació, el Websocket, l'usuari podrà seguir detingudament la simulació que es dur a terme en el servidor.

En la figura 3.7 veiem com l'usuari pot seguir la seva simulació que s'està reproduint en el servidor, i veure per cada element les diferents capes del model TCP/IP que està succeint en cada una de elles. A través d'un quadre de diàleg podem veure totes les capes que té aquell element i analitzar detingudament el recorregut del missatge al llarg de la nostra simulació.

Els quadres de diàleg són únics per cada objecte/canal, o sigui, un objecte connectat a dos canals de comunicació, tindrà dos quadres de diàleg independents, un per cada canal.

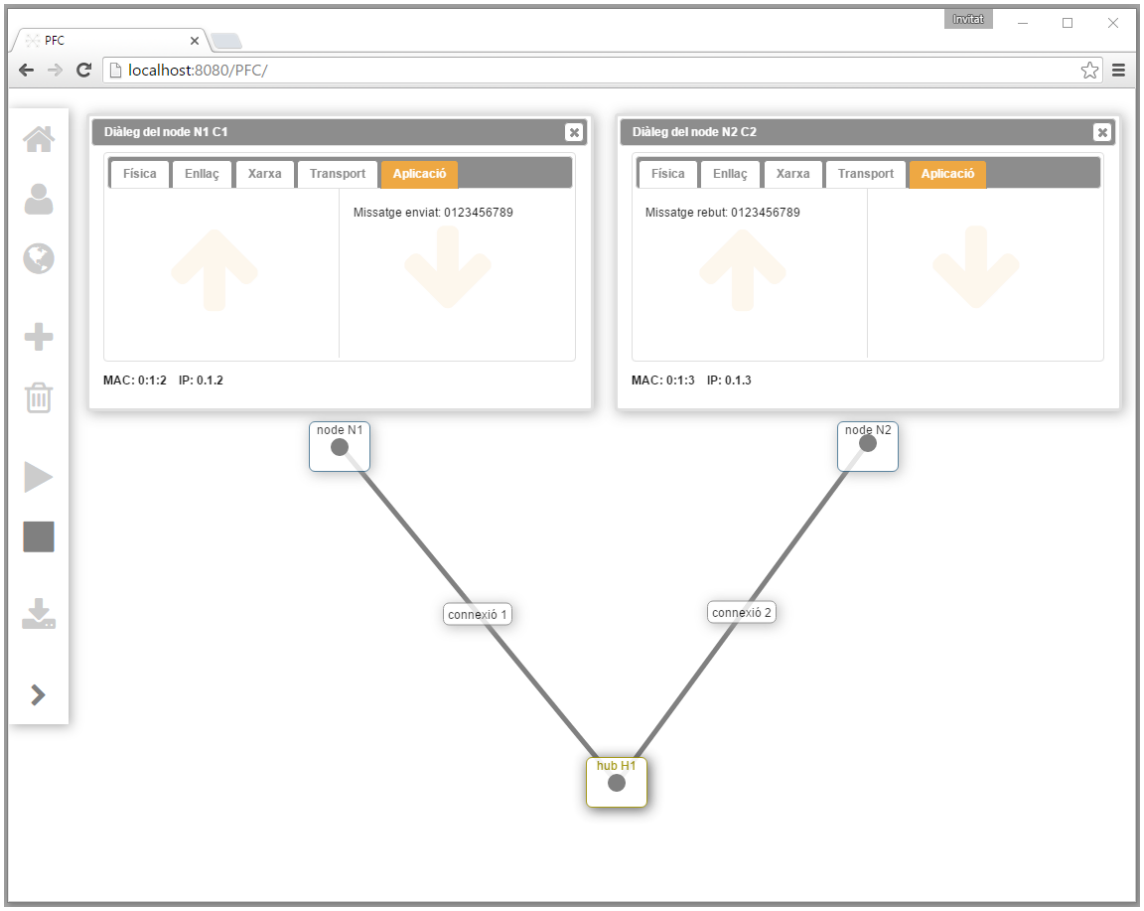

Fig. 3.7 – Entorn web visualització simulació

A més a més podem observar en el quadre de diàleg quina adreça MAC i quina IP posseeix aquell element dins de la xarxa personalitzada. Aquests paràmetres són adjudicats prèviament pel servidor, a través d'un algoritme personalitzat, abans de començar la simulació.

Un altre funcionalitat important del simulador és que ens permet exportar el projecte realitzat per més endavant poder tornar a treballar-hi important la configuració desada. Tindrem dues opcions per desar el nostre projecte: mitjançant un fitxer o través de base de dades.

La importació/exportació mitjançant fitxer, el simulador crea un fitxer de text on desa les configuracions del projecte amb una estructura JSON. En canvi, amb la importació/exportació mitjançant la base de dades, necessitarem

primer identificar-nos com a usuari del simulador i llavors podrem desar la configuració emprada o utilitzar configuracions desades anteriorment.

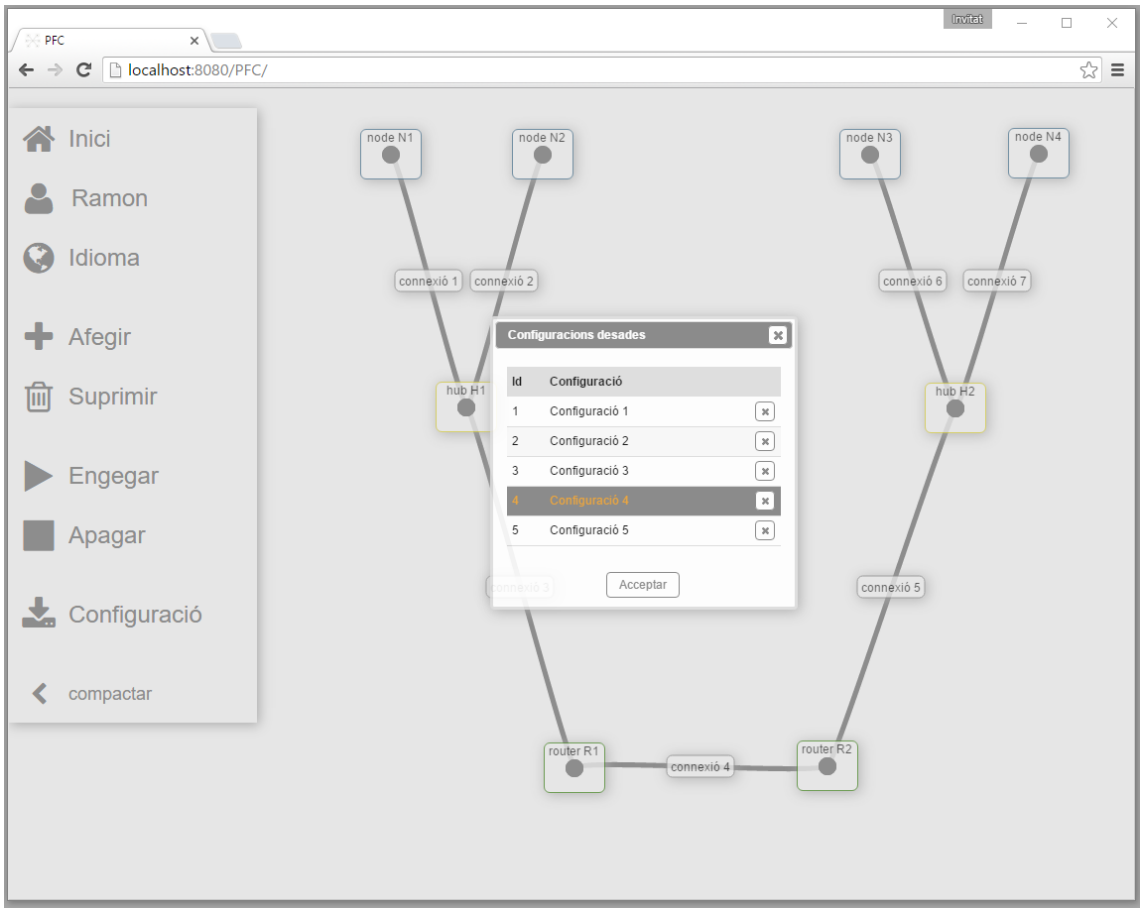

Fig. 3.8 – Entorn web importació simulació

Com hem pogut analitzar l'entorn web del client de l'aplicació té una plataforma fàcil d'utilitzar, simple i amb totes les eines necessàries per realitzar una bona simulació.

Per realitzar tota aquesta interfície gràfica, hem utilitzat un entorn web amb programació HTML5, maquetació del disseny amb CSS3 i amb llibreries i codis de manipulació dels elements amb Javascript. (15) (18) (24)

3.1.2. Entorn Java del servidor

En el servidor tenim un entorn desenvolupat amb JAVA, que farà una simulació de les confeccions que rebi de l'entorn web del client. Aquesta simulació la rep a través d'un fitxer JSON que s'envia pel Websocket i que a través de la creació de diferents fils d'execució simularem aquest disseny de comunicació.

L'estructura de les classes Java de la simulació estan dividides en dos parts: configuració i capes.

En l'apartat de configuració trobem una classe *ServerWebSocket*, que és la principal del programa, on es manté la connexió amb el Websocket del navegador de l'usuari i es fan les connexions servidor/client.

```
@ServerEndpoint(value = "/ServerWebSocket")
public class ServerWebSocket
{
    Session session;
    JSON configurationClient;
    Configuracio configuration;

    @OnOpen
    public void onOpen(Session session) throws IOException {
        System.out.println("Connection OPEN");
    }
    @OnMessage
      public void onMessage(String message, Session session) throws IOException, ParseException {

        JSONObject messageJSON =(JSONObject)new JSONParser().parse(message);
        String messageType=(String) messageJSON.get("type");
        switch(messageType)
        {
            case "CONF":
                JSONObject configurationJSON=(JSONObject) messageJSON.get("configuration");
                Gson gson = new Gson();
                configurationClient = gson.fromJson(configurationJSON.toString(),JSON.class);
                configurationClient.calculaIPsMACs();

                String rConf = "{\"type\":\"CONF\",\"configuration\":"+gson.toJson(configurationClient)+"}";
                session.getBasicRemote().sendText(rConf);
            break;
            case "START":
                String idioma = (String) messageJSON.get("idioma");
                String idSender = (String) messageJSON.get("sender");
                String idReceiver = (String) messageJSON.get("receiver");
                int sendSize = Integer.parseInt((String) messageJSON.get("sendSize"));
                int sendNum = Integer.parseInt((String) messageJSON.get("sendNum"));
                configuration = new Configuracio(session,configurationClient,idioma);
                configuration.realitzaConnexions();
                configuration.startNodes();
                configuration.startSenderReceiver(idSender, idReceiver,sendSize,sendNum);
            break;
        }
      }

    @OnClose
    public void onClose() throws IOException {
        System.out.println("Connection CLOSE");
        if (configuration!=null){configuration.stopNodes();}
    }
}
```

Fig. 3.9 – Classe ServerWebSocket

Com es pot observar en la figura 3.9, la classe ServerWebSocket té tres mètodes que s'executen en els esdeveniments següents: opertura del socket, rebuda de dades i tancament del socket. Concretament en el mètode de rebuda de missatges trobem dos casos:

1. Quan el missatge és del tipus "CONF", ens indica que estem rebent una configuració nova que hem d'analitzar i preparar les adreces IP i MAC per a tots els elements de la simulació.
2. Quan el missatge és del tipus "START", ens indica que ja es pot inicialitzar la configuració rebuda i donar pas a la simulació amb tots els elements personalitzats.

Per tant, en la fase inicial tenim que la classe principal rep un missatge amb la configuració que l'usuari a confeccionat des de l'entorn web i ho rebem amb format JSON. A partir d'aquí, amb una llibreria particular, GSON, podem convertir aquesta cadena JSON amb una classe amb els seus atributs i mètodes. (12) (23)

En aquest punt és on el nostre servidor procedeix a fer l'assignació d'adreces IP i MAC a partir d'un algoritme personalitzat. Per fer d'una manera més senzilla les adreces IP i MAC, que són les adreces de nivell de xarxa i enllaç respectivament, hem confeccionat unes adreces amb 3 grups d'enters. Per tant l'adreça IP seria X.X.X on el primer grup ens marca el router al que està connectat, el segon grup ens marca el hub i l'últim ens marca un identificador dintre el mateix grup d'elements interconnectats. En el cas de les adreces MAC funciona de la mateixa manera, 3 grups d'enters de la següent manera, X:X:X, on igual que abans els grups identifiquen on està connectat aquest element.

Per exemple, si tenim una configuració on tenim 2 routers connectats entre sí, amb un hub connectat a cada router i 2 nodes connectats a cada hub. O sigui, 2 routers, 2 hubs i 4 nodes. El resultant d'adreces que el servidor els hi assigna és:

Element	*Connexió amb*	*MAC*	*IP*
router R1	router R2	1:0:1	1.0.1
router R2	router R1	2:0:1	2.0.1
hub H1	router R1	1:1:1	1.1.1
hub H2	router R2	2:1:1	2.1.1

node N1	hub H1 i router R1	1:1:2	1.1.2
node N2	hub H1 i router R1	1:1:3	1.1.3
node N3	hub H2 i router R2	2:1:2	2.1.2
node N4	hub H2 i router R2	2:1:3	2.1.3

A continuació podeu veure el codi del mètode que calcula les adreces de tots els elements de la simulació:

```
public void calculaIPsMACs()
{
    ArrayList<NodeJSON> parsedNodes = new ArrayList<NodeJSON>();
    Iterator<NodeJSON> rIt = Routers.iterator();
    for(int rNum=1;rIt.hasNext();rNum++)
    {
        NodeJSON r = rIt.next();
        r.insertaIP(rNum+".0.1");
        r.insertaMAC(rNum+":0:1");
        for(int rI=0, rK=0; rI<r.retornaNumConnexions(); rI++)
        {
            if(r.retornaConnexio(rI).tipusConnexio()>0)
            {
                rK++;
                NodeJSON h = retornaHubPerId(r.retornaConnexio(rI).retornaId());
                parsedNodes.add(h);
                h.insertaIP(rNum+"."+rK+".1");
                h.insertaMAC(rNum+":"+rK+":1");
                h.insertaGW(rNum+".0.1");
                for(int hI=0, hK=1; hI<h.retornaNumConnexions(); hI++)
                {
                    if(h.retornaConnexio(hI).tipusConnexio()>0)
                    {
                        hK++;
                        NodeJSON n = retornaNodePerId(h.retornaConnexio(hI).retornaId());
                        n.insertaIP(rNum+"."+rK+"."+hK);
                        n.insertaMAC(rNum+":"+rK+":"+hK);
                        n.insertaGW(rNum+"."+rK+".1");
                    }
                }
            }
        }
    }
    Iterator<NodeJSON> hIt = Hubs.iterator();
    for(int hNum=1;hIt.hasNext();hNum++)
    {
        NodeJSON h = hIt.next();
        if(!parsedNodes.contains(h))
        {
            h.insertaIP("0."+hNum+".1");
            h.insertaMAC("0:"+hNum+":1");
            for(int hI=0, hK=1; hI<h.retornaNumConnexions(); hI++)
            {
                if(h.retornaConnexio(hI).tipusConnexio()>0)
                {
                    hK++;
                    NodeJSON n = retornaNodePerId(h.retornaConnexio(hI).retornaId());
                    n.insertaIP("0."+hNum+"."+hK);
                    n.insertaMAC("0:"+hNum+":"+hK);
                    n.insertaGW("0."+hNum+".1");
                }
            }
        }
    }
}
```

Fig. 3.10 – Classe JSON: mètode calculaIPsMacs

A partir d'aquí, amb la configuració i les adreces generades, construïm la classe *Configuració* on tenim tota l'estructura constituïda en classes. Tenim definides diferents classes per cada un dels elements que poden haver-hi en la simulació:

- La classe *Node* en funció del nivell pot ser un node, un router o un hub.
- La classe *Canal* que representa la connexió entre dos elements.

```
public class Configuracio {

    protected JSON configuracio;
    protected ArrayList<Node> Nodes;
    protected ArrayList<Canal> Canals;
    protected String[][] Connexions;

    protected HashMap<String,HashMap<DireccioIP,String>> taulesIP;
    protected HashMap<String,HashMap<DireccioIP,DireccioMAC>> taulaARP;

    protected Session sClient;
    protected Sortida sortida;
    protected String idioma;
    protected Text textSortida;

    protected Node nodeSender;
    protected Node nodeReceiver;

    protected boolean enMarxa;

    public Configuracio(Session s, JSON conf, String idi)
    {
        int i;
        this.configuracio = conf;
        this.idioma = idi;
        this.taulesIP = configuracio.calculaTaulaIPs();
        this.taulaARP = configuracio.calculaTaulaARP();

        this.Nodes = new ArrayList<Node>();
        for(i=0;i<conf.retornaNumNodes();i++)
        {
            NodeJSON nodeConf = conf.retornaNode(i);
            this.Nodes.add(new Node(nodeConf.retornaId(), nodeConf.retornaNivell(), nodeConf.retornaMAC(), nodeConf.retornaIP()));
        }
        for(i=0;i<conf.retornaNumHubs();i++)
        {
            NodeJSON nodeConf = conf.retornaHub(i);
            this.Nodes.add(new Node(nodeConf.retornaId(), nodeConf.retornaNivell(), nodeConf.retornaMAC(), nodeConf.retornaIP()));
        }
        for(i=0;i<conf.retornaNumRouters();i++)
        {
            NodeJSON nodeConf = conf.retornaRouter(i);
            this.Nodes.add(new Node(nodeConf.retornaId(), nodeConf.retornaNivell(), nodeConf.retornaMAC(), nodeConf.retornaIP()));
        }
        this.Canals = new ArrayList<Canal>();
        for(i=0;i<conf.retornaNumCanals();i++)
        {
            CanalJSON CanalConf = conf.retornaCanal(i);
            this.Canals.add(new Canal(CanalConf.retornaId()));
        }
        this.Connexions = new String[conf.retornaNumConnexions()][2];
        for(i=0;i<conf.retornaNumConnexions();i++)
        {
            String[] connexio = conf.retornaConnexio(i);
            this.Connexions[i] = connexio;
        }
        this.sClient = s;
        this.sortida = new Sortida(s);
        this.textSortida = new Text(idi);
        this.enMarxa = false;
        this.nodeSender=null;
        this.nodeReceiver=null;
    }
```

Fig. 3.11 – Classe Configuració: atributs i constructor

En el constructor de la classe veiem com crea una classe *Node* per cada element que troba en la configuració JSON rebuda i per cada connexió crea una classe *Canal*.

També podem observar, que en el constructor també es fa la crida dels mètodes *calculaTaulaIPs* i *calculaTaulaARP*:

El primer mètode *calculaTaulaIPs* crea les taules necessàries per l'encaminament dels paquets que els routers necessiten. Aquesta taula proporciona al router el canal pel qual ha d'enviar el paquet per a cada IP de destí.

Per exemple, en la situació que teníem anteriorment, 2 routers connectats entre sí, amb un hub connectat a cada router i 2 nodes connectats a cada hub, on les connexions o canals són:

Canal	*Elements connectats*
canal C1	router R1 i router R2
canal C2	router R1 i hub H1
canal C3	router R2 i hub H2
canal C4	hub H1 i node N1
canal C5	hub H1 i node N2
canal C6	hub H2 i node N3
canal C7	hub H2 i node N4

La taula d'encaminament que obtindria el mètode *calculaTaulaIPs* seria la següent:

Router	*Node destí*	*Canal enviament*
router R1	IP node N1	canal C2
router R1	IP node N2	canal C2
router R1	IP node N3	canal C1
router R1	IP node N4	canal C1
router R2	IP node N1	canal C1
router R2	IP node N2	canal C1
router R2	IP node N3	canal C3
router R2	IP node N4	canal C3

El segon mètode *calculaTaulaARP* crea una taula personalitzada per a cada element de la configuració on relaciona cada adreça IP de destí a nivell de xarxa amb una adreça MAC de destí a nivell d'enllaç.

Per exemple, en la situació anterior, la taula ARP del node N1 seria la següent:

Element taula	*IP destí*	*MAC destí*
node N1	IP node N1	MAC node N1
node N1	IP node N2	MAC node N2
node N1	IP router R1	MAC router R1
node N1	IP router R2	MAC router R1
node N1	IP node N3	MAC router R1
node N1	IP node N4	MAC router R1

Com es pot observar tots els elements que no estan a la mateixa xarxa que el node N1 tenen com a MAC de destí el router R1 que és el que encamina tots els paquets de la xarxa a l'exterior d'aquesta.

De la mateixa manera obtindríem la taula ARP dels node N2, N3 i N4 i dels routers R1 i R2. No adjuntem les taules ja que el procediment és el mateix que hem vist pel node N1. Els hubs H1 i H2 al no tenir nivell de xarxa queden fora dels càlculs de les taules ARP.

Tornat a la classe Configuració, aquesta posseeix alguns mètodes importants com són *realitzaConnexions*, *startNodes*, *stopNodes* i *startSenderReceiver*.

El primer mètode *realitzaConnexions* ens crearà la connexió entre els Nodes i els Canals de la nostra simulació, d'aquesta manera tindrem connectats tots els elements de la simulació.

Els mètodes *startNodes* i *stopNodes* ens serviran per engegar o apagar la simulació, ja que cada Node són un conjunt de *Threads* i més endavant analitzarem la torre de protocols que posseeix cada un d'ells.

Per acabar ens queda mencionar el mètode *startSenderReceiver* on el node emissor és el que s'encarrega de començar la simulació enviant els missatges al receptor i a partir d'aquest mètode arrenca la comunicació.

En la figura 3.12 podeu veure el codi d'aquests quatre mètodes que acabem de descriure:

```
public void realitzaConnexions()
{
    int k=0;
    Node node = null;
    Canal canal = null;
    for(int i=0; i<Connexions.length; i++)
    {
        for(k=0;k<Nodes.size();k++)
        {
            if(Nodes.get(k).retornaId().equals(Connexions[i][0])){node=Nodes.get(k);break;}
        }
        for(k=0;k<Canals.size();k++)
        {
            if(Canals.get(k).retornaId().equals(Connexions[i][1])){canal=Canals.get(k);break;}
        }
        if(node!=null && canal!=null){node.enllacarConfiguracio(this);node.assignaCanal(canal);}
    }
}
public void startNodes()
{
    enMarxa=true;
    for(int i=0; i<Nodes.size(); i++)
    {
        Nodes.get(i).start();
    }
    sortida.start();
}
public void stopNodes()
{
    enMarxa=false;
    for(int i=0; i<Nodes.size(); i++)
    {
        Nodes.get(i).stop();
    }
}
public void startSenderReceiver(String idSender, String idReceiver, int sizePaquet, int numPaquet)
{
    for(int i=0; i<Nodes.size(); i++)
    {
        if(Nodes.get(i).retornaId().equals(idSender))nodeSender=Nodes.get(i);
        if(Nodes.get(i).retornaId().equals(idReceiver))nodeReceiver=Nodes.get(i);
    }

    if(nodeSender!=null && nodeReceiver!=null)
    {
        nodeSender.assignaDesti(nodeReceiver);
        nodeSender.sender(sizePaquet,numPaquet);
    }
}
```

Fig. 3.12 – Classe Configuració: mètodes

Per parlar del següent apartat, les capes, hem d'analitzar la classe *Node*. En aquesta classe trobarem el model TCP/IP posat en pràctica. Cada Node té com a atribut una taula amb les diferents capes que aquest element posseeix. En funció del nivell cada element que construïm amb la classe Node pot tenir una taula de capes diferent. En el nostre cas, tenim tres classes d'elements que tenen un nivell de capes diferent i que detallem a continuació:

Element	*Nivell*	*Classes Capa*
Node	5	Física Enllaç Xarxa Transport Aplicació
Router	3	Física Enllaç SuperXarxa
Hub	2	Física SuperEnllaç

Com es pot observar en la taula a cada element se li assigna una taula amb capes. Totes aquestes capes hereten d'una classe abstracta anomenada *Capa*. (10)

En aquest projecte utilitzarem els nivells del model TCP/IP, però desglossarem el primer nivell en els seus dos corresponents al model OSI. Concretament utilitzarem els següents nivells: aplicació, transport, xarxa, enllaç i física. Aquests nivells seran classes que hereten de la classe *Capa*.

A més a més, també hem creat unes classes diferents per als nivells de xarxa i enllaç per als routers i hubs respectivament, ja que aquests es comporten relativament diferent en el seu nivell de màxim ordre. Aquestes classes que en la taula podem observar que són *SuperXarxa* i *SuperEnllaç* hereten d'una variació de la classe *Capa*, que es diu *SuperCapa*.

Tant la classe *Capa* com la classe *SuperCapa* hereten d'una classe superior anomenda *BasicCapa*, que seria la classe origen de tots els tipus de classes de les capes possibles del model TCP/IP.

A continuació farem un anàlisis d'aquestes tres classes origen, que ens servirà per entendre el comportament que tindran totes les capes de la nostra torre de protocols.

```
public abstract class BasicCapa implements Runnable {

    protected Node nodeCapa;

    public abstract void enllacarCapaSuperior(BasicCapa CapaSup);
    public abstract void enllacarCapaInferior(BasicCapa CapaInf);

    public abstract BasicCapa retornaCapaSup();
    public abstract BasicCapa retornaCapaInf();

    public abstract void send(Paquet p);
    public abstract Paquet receive();

    public abstract void Put(Paquet p);

    public void enllacarNode(Node n){nodeCapa=n;}
    public Node retornaNode(){return nodeCapa;}

    public abstract class RunCapa implements Runnable
    {
        protected BasicCapa capa;

        public RunCapa(BasicCapa c)
        {
            this.capa = c;
        }
    }
}
```

Fig. 3.13 – Classe BasicCapa

La classe BasicCapa posseeix els atributs i mètodes principals que faran servir totes les classes dels nivells de la torre TCP/IP. Aquests mètodes són:

- enllacarCapaSuperior i enllacarCapaInferior: els següents mètodes ens serviran per relacionar les capes superior i inferior per tal de poder passar-se la informació.
- send i receive: amb aquests mètodes les capes s'enviaran les dades de la informació a comunicar.
- enllacarNode i retornaNode: els utilitzarem per a que les capes tinguin identificat a quin Node estant treballant.

A més a més la classe és un *Runnable* això vol dir que hi haurà un fil d'execució que durà a terme unes tasques que ha de realitzar la capa en qüestió. Tanmateix la classe posseeix una subclasse *RunCapa*, també abstracta, que és també un Runnable. Aquesta subclasse ens servirà per tenir un fil d'execució per la comunicació de pujada i de baixada, o sigui, la informació que enviem i la que rebem. (10)

```
public abstract class Capa<Up extends Paquet,Down extends Paquet> extends BasicCapa {

    protected String idCanalConnect;

    protected Buffer<Down> bufferDown;
    protected Buffer<Up> bufferUp;

    protected BasicCapa capaInf;
    protected BasicCapa capaSup;

    public abstract void sendInv(Paquet p);
    public abstract Paquet receiveInv();

    public abstract void PutInv(Paquet p);

    protected RunCapa runCapaUp;
    protected RunCapa runCapaDown;


    @Override
    public void enllacarCapaSuperior(BasicCapa CapaSup){capaSup= CapaSup;}
    @Override
    public void enllacarCapaInferior(BasicCapa CapaInf){capaInf= CapaInf;}

    public void enllacarIdCanal(String id){idCanalConnect=id;}
    public String retornaIdCanal(){return idCanalConnect;}

    public abstract class RunCapa implements Runnable
    {
        protected Capa<Up,Down> capa;

        public RunCapa(Capa<Up,Down> c)
        {
            this.capa = c;
        }
    }
    @Override
    public BasicCapa retornaCapaSup(){return capaSup;}
    @Override
    public BasicCapa retornaCapaInf(){return capaInf;}
}
```

Fig. 3.14 – Classe Capa

La classe *Capa* hereta de la classe *BasicCapa* i per tant hereta tots els atributs i mètodes d'aquesta. En aquest cas, les classes Capa disposen d'una capa inferior i una de superior, juntament amb dos *Buffer*, un de pujada i un de baixada, que són unes cues on s'emmagatzemen els paquets que s'intercanvien entre capes. La classe *Buffer*, és un objecte *CircularQueue*, o sigui, una cua circular. Això vol dir que la cua té un número finit d'elements i es van emmagatzemant començant per la primera posició i seguint seqüencialment amb les següents posicions, també de la mateixa manera, es van traient els elements començant per la primera posició i seguint per les següents posicions. La peculiaritat de que sigui circular és quan està escrivint o traient de l'última posició, la següent posició a utilitzar és la primera.

També cal esmentar que la classe CircularQueue està controlada per Monitors. Concretament vol dir que es controla l'accés a aquestes cues, o

sigui que si s'està escrivint o traient un element a la cua, ningú més al mateix moment pot utilitzar la cua, per tant la cua queda bloquejada temporalment. També hem de dir que els Monitors controlen quan la cua està plena o està buida i per tant bloqueja l'escriptura d'elements quan la cua està plena i de la mateixa manera bloqueja l'extracció d'elements quan la cua està buida.

Aquests moments que la cua queda bloquejada els fils d'execució queden esperant a poder realitzar la tasca que volen fer. O sigui, que si tenim per exemple un fil d'execució que vol escriure un element a la cua però aquesta està plena, el fil queda a la espera que s'allibera una posició de la cua i pugui dur a terme la serva tasca.

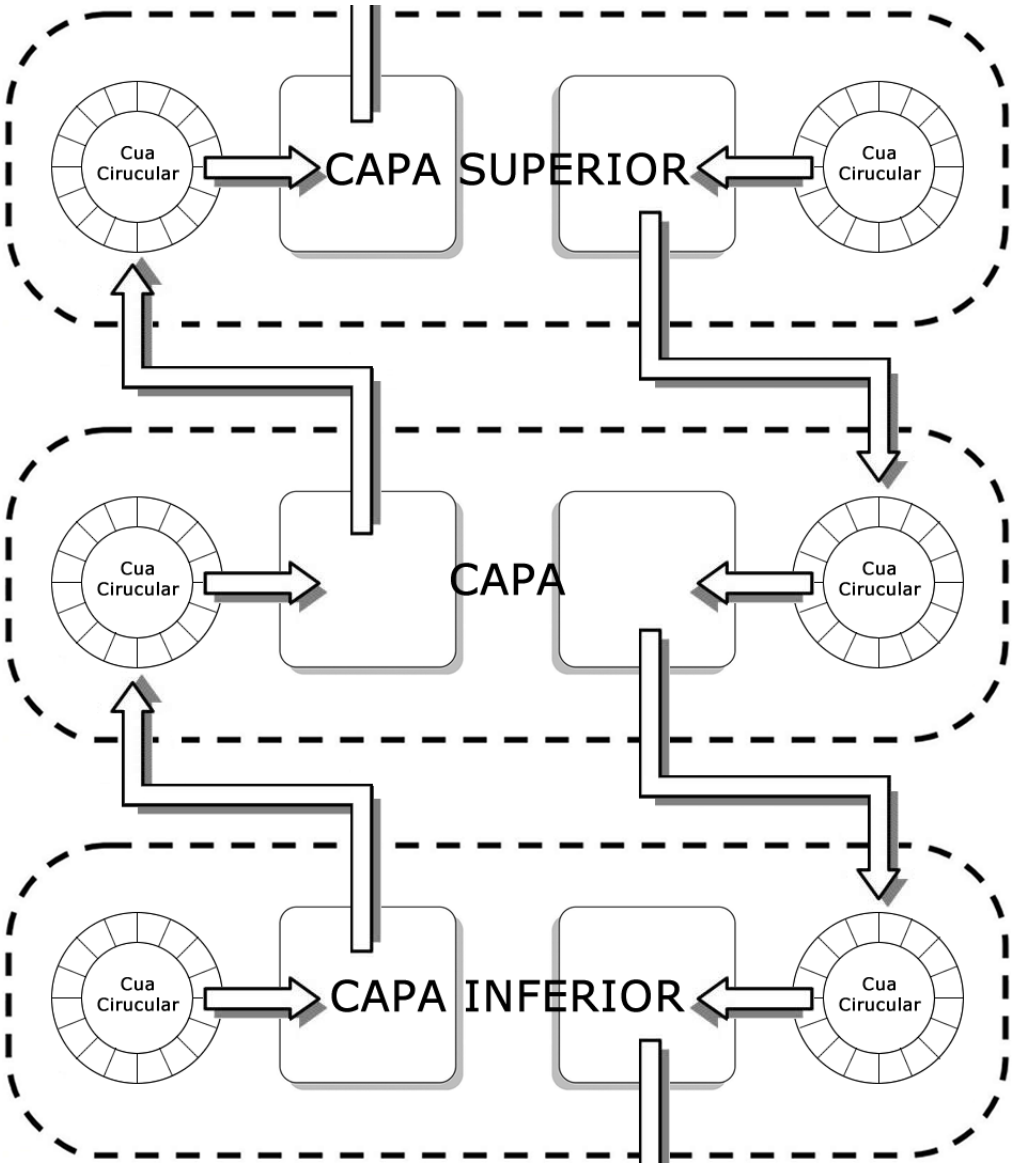

Fig. 3.15 – Diagrama Capa

En el diagrama de la 3.15 veiem l'estructura que segueix la classe Capa on tenim clarament 2 parts diferenciades que serien la d'enviament o baixada i la de rebuda o pujada. Com es pot veure cada Capa té el fil d'execució de baixada, que treu un Paquet del Buffer el processa afegint dades de la capçalera i/o la cua segons el protocol de la seva capa i l'emmagatzema en el Buffer de la capa inferior. En canvi en el fil d'execució de pujada, treu un Paquet del Buffer, el processa traient les dades de la capçalera i/o cua del seu protocol i l'emmagatzema en el Buffer de la capa superior. (8)

```
public abstract class SuperCapa<UpDown extends Paquet> extends BasicCapa {

    protected Buffer<UpDown> buffer;

    protected ArrayList<Capa<? extends Paquet,? extends Paquet>> listCapes;

    protected RunSuperCapa runSuperCapa;

    @Override
    public void enllacarCapaSuperior(BasicCapa Capa){}

    @SuppressWarnings("unchecked")
    @Override
    public void enllacarCapaInferior(BasicCapa Capa)
    {
        listCapes.add((Capa<? extends Paquet, ? extends Paquet>)Capa);
    }

    @Override
    public BasicCapa retornaCapaSup() {return null;}
    @Override
    public BasicCapa retornaCapaInf() {return null;}

    public abstract class RunSuperCapa implements Runnable
    {
        protected SuperCapa<UpDown> supercapa;

        public RunSuperCapa(SuperCapa<UpDown> c)
        {
            this.supercapa = c;
        }
    }
}
```

Fig. 3.16 – Classe SuperCapa

Finalment tenim la classe SuperCapa que també hereta de la classe BasicCapa. És una variació de la classe Capa per als casos especials on tenim vàries connexions al mateix element, com és el cas del hub i del router. En aquests casos la classe Node d'aquests elements tindrà vàries torres de protocols, una per cada connexió i compartiran totes l'última capa (la de nivell superior). Com es pot veure en la figura 3.16, en aquesta classe ara tenim una taula de capes inferiors, i no una de sola com passa a la classe Capa. Aquí és on veiem que aquesta classe SuperCapa distribueix els paquets a les

classes inferiors seguint el protocol que marqui el nivell al que la estem aplicant.

En la figura 3.17 veiem un diagrama de la compartició de la classe SuperCapa amb altres Capes.

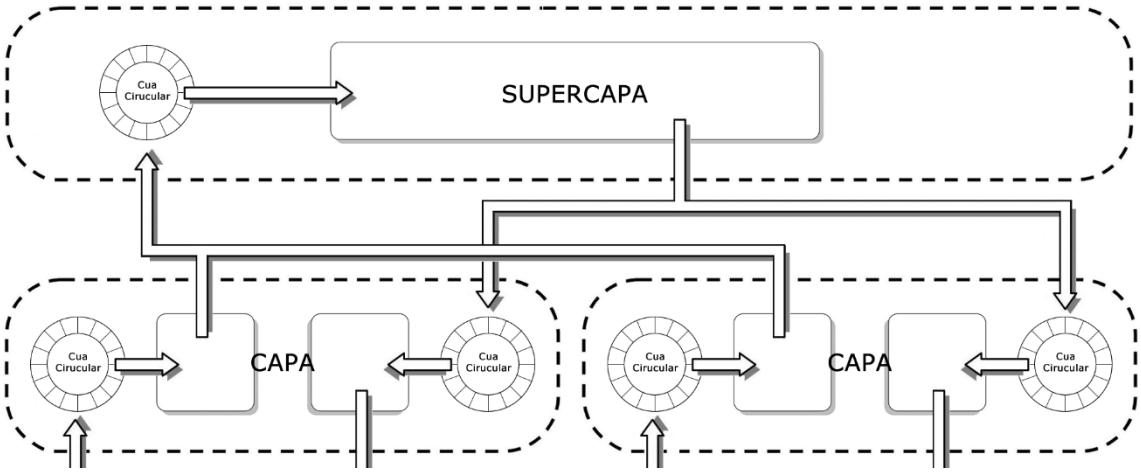

Fig. 3.17 – Diagrama SuperCapa

Un cop vist les tres classe abstractes de les que hereten les nostres classes de capa de cada nivell del model TCP/IP, abans d'explicar quin és el protocol i codi que hem emprat a cada una de elles, primer cal comentar els tipus paquets que viatgen al llarg de la torre de protocols. Concretament el paquet de dades està format pel missatge i un encapsulament amb dades d'informació i de control que utilitzaran cada protocol per a la comunicació. Per tant, quan fem un enviament cada paquet es va encapsulant dintre del següent successivament com es pot observar en la figura 3.18: (1)

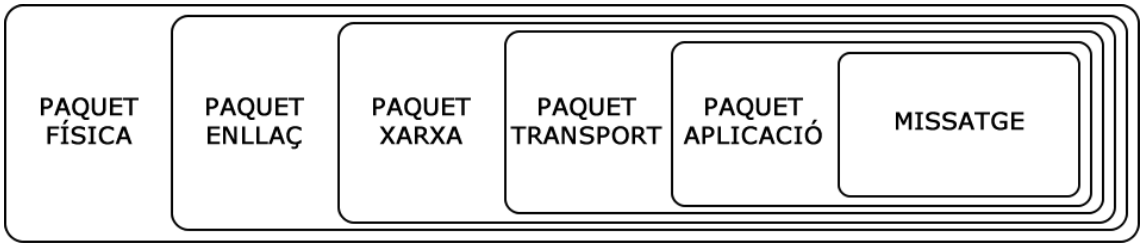

Fig. 3.18 – Diagrama encapsulament paquet

En canvi, en la rebuda el procés és el contrari, cada nivell va desencapsulant els paquet que conté al interior fins a la última capa on s'accedeix al missatge.

Per a fer aquesta estructura on cada paquet té la mateixa composició hem utilitzat una classe abstracte *Paquet* de la qual heretaran tots els tipus de paquets.

A continuació procedirem a explicar les classes dels nivells del model TCP/IP que hem utilitzat per a aquest projecte.

Com ja hem esmentat prèviament, els nivells o capes que hem utilitzat per a crear la torre de protocols són: Física, Enllaç, Xarxa, Transport i Aplicació.

Passem a comentar totes les classes *Capa* i *SuperCapa* que hem desenvolupat per als cinc nivells que acabem d'esmentar:

Física:

Primera part de la primera capa del model TCP/IP. Hereta de la classe *Capa* i per tant també de la classe *BasicCapa*.

Aquest nivell té una singularitat respecte a les altres, i és que al ser la capa més baixa no té cap capa inferior. Sinó que se li assigna el *Canal* amb el qual enviarà o rebrà les trames bit a bit.

La classe *Canal* per garantir que els Nodes connectats a ella poguessin enviar i rebre sense problemes de pèrdua ni de duplicats, té com a atribut una taula de *Buffers*, un per *Node* connectat, on s'emmagatzemarà les trames que la capes *Fisica* enviïn. Per evitar duplicats, quan un *Node* enviï una trama es col·locarà en les cues que té el Canal per als altres *Nodes* (no al Node que envia). Com ja hem explicat abans, al ser cues Buffer, estan controlades per Monitors per evitar tots els tipus de problemes alhora d'introduir o extreure elements d'aquesta.

La trama que utilitza aquesta capa la representem amb la classe *PaquetFisica*. El cos d'aquesta trama seria el paquet de la capa superior, o sigui, la capa enllaç i per tant la classe *PaquetEnllac*.

Enllaç:

Segona part de la primera capa del model TCP/IP. Hereta de la classe *Capa* i per tant també de la classe *BasicCapa*.

En aquesta capa utilitzem el protocol ARP. Aquest protocol ens permet resoldre direccions MAC dels Nodes connectats en el mateix medi. D'aquesta manera tots els Nodes que comparteixen el medi disposen les adreces MAC de tots ells, facilitant la seva comunicació i resoldre possibles errors.

Com ja hem explicat amb anterioritat, es calculen les taules ARP per cada Node al inici de la simulació. Es fa d'aquesta manera per a simplificar l'obtenció d'aquestes, ja que en la realitat les taules es van confeccionant a mesura que es van produint comunicacions.

La taula ARP proporciona una adreça MAC per a una direcció IP de destí donada. Per tant, quan aquesta capa ha d'enviar una trama, col·loca en les capçaleres la MAC origen, que és ell mateix, i la MAC destí, que la obté d'aquesta taula a partir de la IP destí coneguda.

Per contra, en la rebuda d'una trama, aquesta capa és capaç de descartar trames si la MAC de destí que conté la trama no és la seva adreça MAC i que per tant la trama no va destinada al Node en qüestió. En aquest cas la trama serà rebutjada i no s'enviarà al nivell superior. (1)

La trama que utilitza aquesta capa la representem amb la classe *PaquetEnllac* que conté les adreces MAC d'origen i destí de la trama. El cos d'aquesta trama seria el paquet de la capa superior, o sigui, la capa xarxa i per tant la classe *PaquetXarxa*.

SuperEnllaç:

Segona part de la primera capa del model TCP/IP. Hereta de la classe *SuperCapa* i per tant també de la classe *BasicCapa*.

Aquest classe és igualment de nivell d'enllaç però s'utilitza exclusivament per a objectes que siguin hubs. És una classe compartida amb totes les classes *Fisica* que existeixin en el *Node*. En aquest cas, la classe quan rep una trama, l'envia en "broadcast", o sigui, envia la trama a tots els Canals als que està connectat el hub. D'aquesta manera tots els elements connectats amb el hub rebran la trama enviada. Llavors cada un d'ells decidirà si el descarta a nivell d'enllaç si el destí d'aquest no és correcte.

Xarxa:

Segona capa del model TCP/IP. Hereta de la classe *Capa* i per tant també de la classe *BasicCapa*.

En aquesta capa emprarem el protocol IP. Encarregat de crear una connexió, no orientada a la connexió, on es transfereix paquets commutats de manera bidireccional a través de diferents xarxes físiques prèviament enllaçades segons la norma OSI.

La capa enviarà paquets afegint a la capçalera d'aquests l'adreça IP d'origen, que és ell mateix, i la IP de destí on es vol enviar la informació. En canvi, quan es rep un paquet a aquest nivell, extreu l'adreça de destí i sinó és la seva, el descartarà, ja que aquella informació no va dirigida al Node en qüestió. (1)

El paquet que utilitza aquesta capa la representem amb la classe *PaquetXarxa* que conté les adreces IP d'origen i destí del paquet. El cos d'aquesta trama seria el paquet de la capa superior, o sigui, la capa xarxa i per tant la classe *PaquetTransport*.

SuperXarxa:

Segona capa del model TCP/IP. Hereta de la classe *SuperCapa* i per tant també de la classe *BasicCapa*.

Aquest classe és igualment de nivell de xarxa però s'utilitza exclusivament per a objectes que siguin routers. És una classe compartida amb totes les classes *Enllac* que existeixin en el *Node*.

En aques cas, seguirem utilitzant el protocol IP però a més a més, utilitzarem les taules IP d'encaminament que ja hem parlat amb anterioritat. La capa llegirà del paquet la IP de destí i consultarà a aquesta taula per quin canal ha d'enviar el paquet rebut.

Transport:

Tercera capa del model TCP/IP. Hereta de la classe *Capa* i per tant també de la classe *BasicCapa*.

En aquesta capa utilitzem el protocol de finestra lliscant. La finestra lliscant és un mecanisme bidireccional de control de flux. L'emissor envia paquets d'informació i espera la confirmació d'aquests per part del receptor.

La finestra lliscant té un nombre limitat de paquets d'enviament sense confirmació, per tant, sense la validació per part del receptor dels paquets rebuts, l'emissor no pot enviar més informació. D'aquesta manera es realitza el control de flux en la comunicació.

En cas que el receptor enviés una NO validació d'algun paquet o simplement no es rebi validació per part del receptor durant un cert temps, l'emissor procedirà a la retransmissió del paquet. (1)

Per controlar tots els esdeveniments que tindrem en aquest protocol, hem utilitzat la classe *Reactor*. Aquest objecte es basa en crear esdeveniments a partir d'una implementació de la classe *EventHandler*, que es crida quan el *Handle* fa saltar l'esdeveniment. O sigui, l'objecte que implementem de la classe *Handle*, invocarà el seu corresponent esdeveniment *EventHandler* quan es compleixen unes condicions establertes en la classe.

En el nostre cas, utilitzarem varis tipus d'esdeveniments per poder controlar el protocol de finestra lliscant establert en aquesta capa:

- Temporitzadors: per poder realitzar un temps d'espera a que arribi la validació per part del receptor del paquet enviat. Cada paquet que enviem i que es manté en la finestra lliscant, crea un temporitzador per poder fer la retransmissió en cas de no rebre la validació o de rebre una NO validació per part del receptor.
- Paquets en cua: quan les cues Buffer de la capa reben un element, es faria la crida d'aquest esdeveniment. D'aquesta manera estarem alerta de quan les capes superior i inferior ens dipositen paquets en les cues.

Per realitzar aquestes operacions la classe *Handle*, una classe abstracta amb un mètode *ready* a implementar, ens retornarà un booleà en funció del codi que emprem en aquest mètode. Llavors, es faria la crida del mètode de la classe abstracte a implementar *EventHandler*, el mètode *handle_event,* on aquesta funció ens permetria realizar una acció.

El Reactor és l'element on registrem un Handle amb el seu corresponent EventHandler, que serien la condició a complir i l'execució a dur a terme un cop es compleixi la condició.

De la mateixa manera que registrem aquesta parella d'elements en el *Reactor*, també en qualsevol moment els podem eliminar. Aquest seria el cas dels temporitzadors, ja que la condició d'aquests és un compte enrere que quan s'esgota, executaria l'esdeveniment. Per tant, quan la finestra lliscant rep la validació per part del receptor, elimina el temporitzador del *Reactor*. (8)

En la figura 3.19 podem veure l'estructura d'aquests tres elements necessaris que utilitza el Reactor.

```
public class Reactor {

    private ReactorImplementation reactor_impl;

    public Reactor() {
    reactor_impl = ReactorImplementation.instance();
    }

    public void register_handler(Handle h, EventHandler eh) {
    reactor_impl.register_handler(h,eh);
    }

    public void remove_handler(Handle h) {
    reactor_impl.remove_handler(h);
    }

    public void handle_events(int milisecs) {
    reactor_impl.handle_events(milisecs);
    }
    public void notify_event(Handle h){
    reactor_impl.notify_event(h);
    }
}

public interface Handle {

    public boolean ready();

}
public interface EventHandler {

    public void handle_event(Handle h);

}
```

Fig. 3.19 – Classe Reactor, Handle i EventHandler

El paquet que utilitza aquesta capa la representem amb la classe *PaquetTransport* que conté els següents paràmetres:

- Tipus de paquet, si és data, ack o nack. Data quan conté informació a enviar; ack quan és un paquet de validació de rebuda; nack quan és un paquet de no validació a causa d'algun error.
- Número de seqüencia: és un número que permet al receptor ordenar els paquets que va rebent, ja que no té perquè rebre'ls de forma seqüencial.
- Número Ack: és el número de seqüencia que el receptor valida o no en funció del tipus de paquet que s'indica.

El cos d'aquesta trama seria el paquet de la capa superior, o sigui, la capa aplicació i per tant la classe *PaquetAplicacio*. El cos d'aquesta trama seria el paquet de la capa superior, o sigui, la capa xarxa i per tant la classe *PaquetTransport*.

Aplicació:

Quarta capa del model TCP/IP. Hereta de la classe *Capa* i per tant també de la classe *BasicCapa*.

El paquet que utilitza aquesta capa la representem amb la classe *PaquetAplicació* que conté els ports d'origen i de destí que utilitza l'aplicació. El cos d'aquest paquet seria el missatge que el Node emissor confecciona.

Per a fer una simulació d'un enviament per part del Node que definim com a emissor, es crea un *Array* de bytes amb un número finit de posicions. El número de paquets i el número de posicions a enviar és definit per l'usuari en la interfície gràfica de l'aplicació.

Finalment el Node que hem definit com a receptor rep els missatges que s'han enviat per part del Node emissor.

Base de dades

Un altre punt important a destacar en l'apartat del servidor és la base de dades MySQL. Apache Tomcat connecta a través d'un connector JDBC amb la base de dades i l'aplicació pot emmagatzemar i consultar la informació sobre els usuaris i les configuracions que aquests exporten. (13)

A continuació farem un anàlisi de les taules que hem emprat a la base de dades i els Servlets que hem creat per a que l'entorn web de l'aplicació pugui consultar aquesta informació.

Les taules MySQL que hem utilitzat per al nostre projecte són les següents:

id	user	passwd	propietari	actiu
1	user 1	password 1	Usuari 1	Y
2	User 2	password 2	Usuari 2	Y

La taula que acabem de veure és la taula usuaris on podem observar que hi tenim un identificador, l'usuari, el password i un camp actiu que és un enum amb valor Y (yes) i N (no).

La segona taula que utilitzem en la nostra base de dades és la taula configuracions, on emmagatzemem les simulacions que els usuaris vulguin guardar. La configuració la guardarem amb la cadena JSON que utilitzem després per a la importació.

id	id_user	descripció	configuració
1	1	Configuració 1	{.....}
2	1	Configuració 2	{.....}

En aquesta segona taula observem la relació amb la primera taula a través del camp id_user, que fa referència al camp identificador de la primera.

Un cop vist les dues taules que tenim a la nostra base de dades, analitzem els dos Servlets que posseeix el servidor per a que l'entorn web del client pugui consultar directament la informació desada.

Aquest dos Servlets són les dues classes *Users* i *UsersConf*. Anem a fer un petit anàlisi de cada una de elles.

Aquestes classes hereten de *HttpServlet*, objecte abstracta que té els mètodes *doGet* i *doPost*. Aquests dos mètodes responen a les peticions GET i POST del protocol HTTP. En el nostre cas, els dos mètodes seran idèntics ja que volem que responguin de la mateixa manera amb les dues formes de petició.

Les dues classes funcionen de la mateixa manera, responen a la petició retornant un XML. Des de l'entorn web del client es fa una crida al servidor, en el nostre cas totes seran per GET, demanant informació sobre els usuaris o sobre les simulacions desades i els HttpServlet implementats retornen una cadena XML amb la informació que es demana. (25)

El primer d'aquests Serlets és la classe *Users*. Aquesta classe és la encarregada de contestar totes les peticions referents als usuaris de l'aplicació. Tenim varis tipus de petició:

- Login: Petició que comprova si el usuari i contrasenya introduïts són correctes. Retorna un XML marcant si és correcte el login i si és correcte retorna totes les dades d'aquest usuari i el guarda en variables de sessió per a quan es facin altres peticions pugui recuperar les seves dades.
- Logout: Petició que tanca la sessió de l'usuari que anteriorment s'hagi autentificat correctament. Per a fer aquesta petició, s'eliminaran les dades de les variables de sessió referents a l'usuari.
- Register: Petició que demana per crear un usuari nou. Comprova que l'usuari demanat no existeixi i si tot és correcte a més a més també fa l'acció de Login i per tant guarda les dades de sessió i les retorna.

El segon Servlet és la classe UsersConf. Aquesta és la classe encarregada de contestar totes les peticions referents a les simulacions o configuracions que els usuaris generen. Les peticions que rep aquesta classe són les següents:

- List: Petició que retorna totes les simulacions desades a la base de dades referent a l'usuari autentificat. Concretament retorna

l'identificador, la descripció i la configuració JSON de cada una de les simulacions.

- Export: Petició que es demana quan es vol fer una exportació per part de l'usuari en l'entorn web de l'aplicació. El Servlet inserta la configuració exportada a la base de dades.
- Delete: Petició que elimina la simulació amb l'identificador que envia l'usuari.

3.2. Implementació de les tecnologies

A continuació analitzarem com hem implementat les tecnologies més importants que hem utilitzat en el nostre projecte. Per fer l'anàlisi classificarem les tecnologies emprades en l'entorn web del client i les utilitzades en l'entorn Java del servidor.

3.2.1. Entorn Web del client

En l'apartat del client, com ja hem dit amb anterioritat, està realitzat amb HTML5 i Javascript. Concretament hem utilitzat les següents llibreries de Javascript que ens ajudaran a crear un entorn gràfic més visual i més simple d'utilitzar: (15) (18)

```
<!DOCTYPE html>
<html>
<head>
<title>PFC</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=Edge">

<link rel="shortcut icon" href="favicon.ico">
<link rel="stylesheet" href="css/jquery-ui-1.10.4.custom.min.css">
<link rel="stylesheet" href="css/font-awesome/css/font-awesome.min.css">
<link rel="stylesheet" href="css/styles.css">

<script type="text/javascript" src="js/jquery-1.9.1.min.js"></script>
<script type="text/javascript" src="js/jquery-ui-1.10.4.custom.min.js"></script>
<script type="text/javascript" src="js/i18next-1.7.3.min.js"></script>
<script type="text/javascript" src="js/jquery.jsPlumb-1.7.5-min.js"></script>
<script type="text/javascript" src="js/Blob.js"></script>
<script type="text/javascript" src="js/FileSaver.js"></script>
<script type="text/javascript" src="js/functions.js"></script>
</head>
```

Fig. 3.20 – Capçalera de la pàgina web principal

Com es pot veure en la figura 3.20 en la pàgina web principal de l'entorn web del client, s'inclouen en la capçalera les llibreries Javascript que analitzarem a continuació i comentarem com les utilitzem en l'entorn web:

JQuery:

És un biblioteca que permet simplificar la manera d'interaccionar amb els documents HTML i manipular els seus objectes, esdeveniments i animacions.

En la nostra aplicació utilitzarem aquesta llibreria per a totes les funcions que hem emprat amb Javascript, ja que és una llibreria bàsica per a la interacció d'elements HTML. (20)

JsPlumb:

És una biblioteca que permet realitzar connexions entre objectes. Ens permet controlar els objectes, esdeveniments i animacions de les connexions i objectes. (19)

```
jsPlumb.ready(function(){
    jsPlumb.importDefaults({
        Connector : [ "Bezier", { curviness:10 } ],
            DragOptions : { cursor: "pointer", zIndex:2000 },
            PaintStyle : { strokeStyle:"gray", lineWidth:5 },
            EndpointStyle : { radius:9, fillStyle:"gray" },
            HoverPaintStyle : {strokeStyle:"#ec9f2e" },
            EndpointHoverStyle : {fillStyle:"#ec9f2e" },
            Anchors :  [ "Center", "Center" ],
            ConnectionOverlays:[[ "Label", { label: i18n.t("objects.connection"),
                id: "label", cssClass: "label" }]],
            Container: "dashboard"
    });
});
```

Fig. 3.21 – Declaració objecte connector de la llibreria jsPlumb

En la figura 3.21 veiem la declaració de l'objecte *Connector* que ens permetrà crear els element Node, Hub, Router de la plataforma. A partir d'aquests connectors es poden realitzar les connexions entre ells a través de "drag and drop", un concepte molt utilitzat en HTML. La llibreria també ens deixa parametritzar tots els esdeveniments que succeeixin amb aquest connectors. Per exemple quan creem una connexió, al clicar per començar a fer la connexió o quan deixem anar per enllaçar la connexió. Tenim una varietat d'esdeveniments que pot generar l'usuari quan utilitza els connectors, que nosaltres podem personalitzar al nostre gust per a que realitzi les accions pertinents.

Un exemple de parametrització d'un esdeveniment seria quan no permeten la connexió de tots els elements. Quan intentem connectar dos nodes entre sí, l'aplicació no ens ho deixa fer. Bé aquest fenomen el podem controlar gràcies a la parametrització d'un esdeveniment de la llibreria.

```
function jsPlumbBind(active)
{
    if(active)
    {
        jsPlumb.bind('connection', function(info){
            var delete_connection = false;
            if(comprovaConnection(info)){preparaConnection(info);}
            else{alertBox(i18n.t("alerts.error_connection"));delete_connection=true;}
            info.connection.setParameter('delete',delete_connection);
        });
        jsPlumb.bind('connectionDragStop',function(connection){
            if(connection.getParameter('delete'))jsPlumb.detach(connection);});
        jsPlumb.bind('connectionDetached', function(connectionInfo){
            if(!connectionInfo.connection.getParameter('delete'))deleteConnection(connectionInfo);
        });
        jsPlumb.bind('dblclick', function (connection,e){jsPlumb.detach(connection);});
    }
    else
    {
        jsPlumb.unbind('connection');
        jsPlumb.unbind('connectionDragStop');
        jsPlumb.unbind('connectionDetached');
        jsPlumb.unbind('dblclick');
    }
}
```

Fig. 3.22 – Esdeveniments del connector de la llibreria jsPlumb

JQuery UI:

És una biblioteca que permet realitzar diferents objectes de disseny de HTML com poden ser els botons, els quadres de diàleg, tabulacions, barres de progrés, etc.

En el nostre cas utilitzarem els quadres de diàleg juntament amb les tabulacions. Per utilitzar aquest objectes, s'han invocar a través de Javascript, es defineix l'objecte HTML i les propietats que tindrà l'objecte diàleg o tabulacions. A continuació, en la figura 3.23 podem veure un exemple de crida d´un objecte "tabs" i un objecte "dialog" on té unes propietats pautades que nosaltres podem variar al nostre gust. (21)

```
$("#tabs"+id+canal).tabs();

$("#dialogTabs"+id+canal).dialog({
    autoOpen:false,
    width:w,
    height:300,
    minWidth:w,
    minHeight:300,
    position:{my:"center bottom", at:"center+"+posicio+" top-"+(posicio+10), of:$("#"+id)},
    open:function(){$("#tabs"+id+canal).tabs("option","heightStyle","fill");},
    resize:function(){$("#tabs"+id+canal).tabs("option","heightStyle","fill");}
});
```

Fig. 3.23 –Objecte tabs i dialog llibreria jQuery UI

I18next:

És una biblioteca que ens permet fer que la pàgina web sigui amb varis idiomes. A través d'unes etiquetes que definim i utilitzem en el codi HTML, la llibreria s'encarrega de reemplaçar aquestes etiquetes per la seva corresponent traducció guardada amb fitxers JSON, prèviament definits. (22)

```
"buttons":
{
    "home": "Inici",
    "user": "Usuari",
    "language": "Idioma",
    "add": "Afegir",
    "delete": "Suprimir",
    "edit": "Modificar",
    "configuration": "Configuració",
    "file": "Fitxer",
    "db": "Base de dades",
    "start": "Engegar",
    "stop": "Apagar",
    "send": "Enviar",
    "compact": "compactar",
    "add_node": "Afegir Node",
    "add_hub": "Afegir Hub",
    "add_router": "Afegir Router",
    "delete_object": "Suprimir l'objecte seleccionat",
    "delete_connection": "Suprimir la connexió seleccionada",
    "delete_objects": "Suprimir tots els objectes",
    "delete_connections": "Suprimir totes les connexions",
    "export_configuration": "Exportar configuració",
    "import_configuration": "Importar configuració",
    "start_configuration": "Engegar configuració",
    "stop_configuration": "Apagar configuració",
    "accept": "Acceptar",
    "cancel": "Cancelar"
},
```

```
<li>
    <a href="javascript:void(0);"><i class="fa fa-plus fa-3x"></i><span class="fa-2x" data-i18n="b
    <ul>
        <li><a href="javascript:void(0);" id="addNode" data-i18n="buttons.add_node"></a></li>
        <li><a href="javascript:void(0);" id="addHub" data-i18n="buttons.add_hub"></a></li>
        <li><a href="javascript:void(0);" id="addRouter" data-i18n="buttons.add_router"></a></li>
    </ul>
</li>
```

Fig. 3.24 – Fitxer JSON i crida HTML de la llibreria i18n

En la figura 3.24 tenim talls de codi tant del fitxer JSON d'un idioma en concret com de la crida HTML per a que la llibreria intercanviï amb la traducció corresponent. En aquest exemple veiem com es poden fer agrupacions d'etiquetes, com seria el cas de "buttons" on engloba vàries etiquetes que fan referència als botons de l'aplicació. Una etiqueta d'aquestes seria "add_node" on veiem com després en el codi HTML fem referència a ella invocant-la a través d'un atribut de l'element que vol etiquetar. L'atribut és el "data-i18n" i el valor d'aquest el referenciem a l'etiqueta "buttons.add_node". D'aquesta manera a l'inici de la càrrega de la pàgina

web, la llibreria substituirà totes les etiquetes que trobi en el document HTML per les corresponents traduccions del fitxer JSON. (23)

A més a més, en termes de HTML5, hem utilitzat dos tecnologies importants:

WebSocket:

És una API de HTML5 que estableix un socket de connexió entre el navegador web i el servidor. Com es pot veure en el codi que hi ha a continuació, el Websocket disposa d'una funció per realitzar la connexió, la funció connect, que comprova que es pugui realitzar la comunicació i que el navegador emprat tingui habilitada aquesta API. També posseeix una funció per desconnectar aquesta comunicació, la funció disconnect. Com també té la funció per enviar missatges al servidor a través de la connexió establerta, la funció sendMessage. (16)

```
var webSocket = {
    ws : null,
    connect : function(conf){
        var URL = http_server.replace("http","ws")+"/PFC/ServerWebSocket";
        if ('WebSocket' in window) {
            this.ws = new WebSocket(URL);
        } else if ('MozWebSocket' in window) {
            thi.ws = new MozWebSocket(URL);
        } else {
            alertBox(i18n.t("alerts.browser_not_supported"));
            return;
        }
        this.ws.onopen = function(){startApplication();webSocket.ws.send(conf);};
        this.ws.onclose = function(){alertBox(i18n.t("alerts.close_websocket"));stopApplication();};
        this.ws.onerror = function (event){alertBox(i18n.t("alerts.error_websocket"));};
        this.ws.onmessage = function(event){
            msg=$.parseJSON(event.data);
            switch(msg.type)
            {
                case "CONF":
                    importJSON(msg.configuration);
                    preparaTabsDialog();
                    webSocket.sendMessage('{"type":"START", "idioma":"'+i18n.lng()+'", "sender":"'+sender+
                            '", "receiver":"'+receiver+'", "sendSize":"'+sendSize+'", "sendNum":"'+sendNum+'"}');
                break;
                case "TEXT":
                    insertTEXT(msg.configuration);
                break;
                default:
                    alertBox("alerts.error_websocket");
                break;
            }
        };
    },
    disconnect : function ()
    {
        if (this.ws != null)
        {
            this.ws.close();
            this.ws = null;
        }
    },
    sendMessage : function (message)
    {
        if (this.ws != null)this.ws.send(message);
    }
};
```

Fig. 3.25 – HTML Websocket

FileReader:

És una API de HTML5 que ens permet llegir un fitxer determinat. En el nostre cas, utilitzem aquesta llibreria per la importació la configuració o simulació realitzada amb el navegador. (17)

```
fileInput = document.getElementById('fileInput');
fileInput.addEventListener('change', function(e) {
    var file = fileInput.files[0];
    var textType = /text.*/;

    if (file.type.match(textType))
    {
        if(isBrowserSupported)
        {
            var reader = new FileReader();
            reader.onload = function(e)
            {
                importFile(reader.result);
            };
            reader.readAsText(file);
        }
        else alertBox(i18n.t("alerts.browser_not_supported"));
    }
    else alertBox(i18n.t("alerts.file_not_supported"));

});
```

Fig. 3.26 – HTML FileReader

Per complementar aquesta funció hem utilitzat dos llibreries conjuntes FileSaver i Blob que ens ajudaran a fer el procés contrari, a fer l'exportació de la configuració, per tant a fer l'escriptura del fitxer. Generarem un fitxer de text, on guardarem la informació necessària dels elements i les connexions realitzades per després poder tornar-ho a reconstruir.

A continuació en la figura 3.27 podem veure el codi de les funcions emprades per a fer la importació i l'exportació d'aquesta configuració.

```
function exportFile()
{
    json=exportJSON(true);
    if(json)
    {
        if(isBrowserSupported)
        {
            blob = new Blob([json],{type: "text/plain;charset=utf-8"});
            saveAs(blob,"configuration.txt");
        }
        else alertBox(i18n.t("alerts.browser_not_supported"));
    }
    else alertBox(i18n.t("alerts.no_connections"));
}
function importFile(inputFile)
{
    inputJSON=$.parseJSON(inputFile);
    if(inputJSON.type=="CONF")
    {
        resetAll();
        jsPlumb.deleteEveryEndpoint();
        importJSON(inputJSON.configuration);
        importDraw(inputJSON.positions);
    }
    else alertBox(i18n.t("alerts.file_incorrect"));
}
```

Fig. 3.27 – Funcions importFile i exportFile

3.2.2. Entorn Java del servidor

En l'apartat del client, tenim un servidor Apache Tomcat i el codi implementat està realitzat amb llenguatge Java. Les llibreries emprades per realitzar la simulació de la xarxa de comunicació són:

websocket-api.jar:

Llibreria inclosa en el nostre projecte que ens serveix per realitzar la connexió amb el Websocket del client. (11)

A continuació podem veure el codi de la classe *ServerWebSocket*, on podem observar els tres mètodes bàsics per realitzar la connexió amb el client: onOpen, onMessage, onClose:

```
@ServerEndpoint(value = "/ServerWebSocket")
public class ServerWebSocket
{
	@OnOpen
	public void onOpen(Session session) throws IOException {...}
	@OnMessage
	public void onMessage(String message, Session session) throws
IOException, ParseException {...}
	@OnClose
	public void onClose() throws IOException {...}
}
```

Fig. 3.28 – Classe ServerWebSocket

Amb els tres mètodes de la classe podem escoltar les peticions emprades pels navegadors dels clients.

gson.jar:

Llibreria Google JSON que ens permet convertir un objecte JSON amb una objecte amb tots els atributs. Ens servirà de gran utilitat, ja que ens transformarà directament la configuració escrita amb una cadena de text a una classe amb tots els objectes com a atributs d'aquesta. (12)

A continuació, podem veure un exemple de cadena amb JSON de la configuració que envia el navegador de l'usuari al servidor i la invocació que fem a la llibreria per obtenir la classe resultant: (23)

```
{
"Connexions":[["N1","C1"],["H1","C1"],["N2","C2"],["H1","C2"]],
"Routers":[],
"Canals":[{"id":"C1"},{"id":"C2"}],
"Relacions":[["N1","H1"],["N2","H1"]],
"Hubs":[{"nivell":2,"id":"H1","connexions":[{"id":"N1","tipus":1},{"id":"N2",
"tipus":1}]}],
"Nodes":[
	{"nivell":4,"id":"N1","connexions":[{"id":"H1","tipus":-1}]},
	{"nivell":4,"id":"N2","connexions":[{"id":"H1","tipus":-1}]}
	]
}
```

```
Gson gson = new Gson();
configuration=gson.fromJson(configurationJSON.toString(),JSON.class);
```

Fig. 3.29 – Cadena JSON configuració i utilització GSON

En aquest exemple de la figura 3.29 veiem que la configuració té dos nodes connectats a un hub. Per tant tenim tres elements (dos nodes i un hub) i dos connexions o canals.

A través de la llibreria GSON, passant-li la cadena JSON i la classe del objecte a transformar, en aquest cas la classe JSON. Obtindrem directament tota la configuració en una sola classe i diferents subclasses i atributs.

A continuació, en la figura 3.30, veiem l'estructura resultant de transformar la cadena JSON en una classe:

```
public class JSON {

        public class NodeJSON
        {
                public class ConnexioNode
                {
                        @SerializedName("id")
                        protected String id;
                        @SerializedName("tipus")
                        protected int tipus;

                        ...
                }
                @SerializedName("id")
                protected String id;
                @SerializedName("nivell")
                protected int nivell;
                @SerializedName("connexions")
                protected ArrayList<ConnexioNode> connexions;

                protected DireccioIP IP;
                protected DireccioMAC MAC;
                protected DireccioIP Gateway;

                ...
        }
        public class CanalJSON
        {
                @SerializedName("id")
                protected String id;

                ...
        }
        @SerializedName("Nodes")
        protected ArrayList<NodeJSON> Nodes;
        @SerializedName("Hubs")
        protected ArrayList<NodeJSON> Hubs;
        @SerializedName("Routers")
        protected ArrayList<NodeJSON> Routers;
        @SerializedName("Canals")
        protected ArrayList<CanalJSON> Canals;
        @SerializedName("Connexions")
        protected String[][] Connexions;
        @SerializedName("Relacions")
        protected String[][] Relacions;

        ...
}
```

Fig. 3.30 – Estructura classe JSON

En la classe JSON veiem com estan etiquetats alguns del atributs de la classe que corresponen als atributs de la cadena JSON que envia el client.

Com veiem en l'estructura, tenim subclasses dintre la pròpia classe. Aquestes serien les classes: NodeJSON i CanalJSON, que són els objectes que trobem en els Array *Nodes*, *Hubs*, *Routers* i *Canals* de la cadena de text entrant.

És important remarcar que en la classe JSON hem marcat cada un dels objectes amb l'etiqueta *@SerializedName("nom")* que li permet a la llibreria GSON relacionar els objectes de la classe amb els objectes de la cadena JSON.

mysql.jdbc.jar:

Llibreria inclosa en el notre projecte que ens permet realitzar la connexió amb una base de dades MySQL. El servidor utilitza una base de dades per a emmagatzemar dos tipus d'informació: els usuaris de l'aplicació amb els seus corresponents credencials i les configuracions desades que l'usuari vulgui exportar. (13)

```
public class MysqlDB {

    protected Connection conn;

    public void open() throws SQLException
    {
        DriverManager.registerDriver(new com.mysql.jdbc.Driver());
        conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/pfc","user","password");
    }
    public ResultSet select(String query) throws SQLException
    {
        Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery(query);
        return rs;
    }
    public void update(String query) throws SQLException
    {
        Statement st = conn.createStatement();
        st.executeUpdate(query);
    }
    public void close() throws SQLException{conn.close();}
}
```

Fig. 3.31 –classe MysqlDB

En la figura 3.31, veiem la classe MysqlDB, l'objecte encarregat de crear la connexió amb la base de dades. Com es pot observar posseeix quatre mètodes que ens permetran realitzar tota les operacions amb aquesta.

En primer lloc tenim els mètodes *open* i *close* que ens serviran per obrir i tancar respectivament la comunicació amb la base de dades. Com es pot

observar en el mètode d'obertura, utilitzem específicament la llibreria que hem comentat amb anterioritat que utilitza el connector jdbc per a crear la comunicació.

Per realitzar totes les consultes a la base de dades utilitzarem els dos altres mètodes *select* i *update*. Amb el primer podrem realitzar totes les consultes SELECT i en canvi, amb l'altre mètode l'utilitzarem per fer INSERT, UPDATE i DELETE. La diferència bàsica entre els dos mètodes és per a les consultes SELECT necessitem el resultat que ens retorna, en canvi per a les altres invocacions no necessitem el retorn.

4.Línies futures

Les línies de continuïtat del nostre projecte les podem classificar en modificacions que es podrien fer al projecte un cop acabat i les noves funcionalitats que es podrien afegir a la nostra aplicació.

Respecte a les modificacions que un cop finalitzat el projecte es podrien realitzar a l'aplicació, podríem destacar la millor visualització en l'entorn web de com viatgen els paquets al llarg de la xarxa. O sigui, buscar un mètode on es pugui veure amb més facilitat quin és el recorregut per on han viatjat els paquets de la comunicació.

A més a més també es podria modificar la visualització del quadres de diàleg on es visualitza la informació dels protocols emprats en el model TCP/IP. El problema ve donat quan aquest element està connectat a un elevat nombre de canals, ja que es genera un diàleg per a cada un d'ells i això complica la comprensió de la seva tasca dintre la simulació.

En quan a les noves funcionalitats que podríem afegir al nostre projecte, destaquem la possibilitat d'incorporar un sistema per a realitzar errors dintre la nostra comunicació. O sigui, que els nostres canals tinguessin la possibilitat de perdre paquets i així comprovar el funcionament del control d'errors per part d'alguns protocols i poder veure la retransmissió d'aquets.

Aquest sistema de pèrdues es podria generar fent que els canals descartessin paquets aleatòriament. Fet que produiria errors en la comunicació i el conseqüent control d'aquests activant la retransmissió dels paquets perduts. A més a més, treballar amb un sistema de visualització d'aquestes pèrdues ja que l'eina està pensada per un entorn didàctic i també poder donar la possibilitat d'escollir quins canals generen pèrdues i quins no.

Una altra nova funcionalitat, i que es simplifica en el nostre projecte, és el càlcul de les taules ARP i de les taules d'encaminament IP. Aquestes en la nostra aplicació es calculen a l'inici de la configuració, en canvi, per fer una simulació que sigui el més real possible, aquestes taules s'haurien d'anar

emplenant a mesura que la comunicació va fluint. Aquesta manera els nodes anirien emmagatzemant aquesta informació a mesura que es vagin necessitant aquestes dades. Per tant aquestes taules inicialment estarien buides i a mesura que avança la simulació es van alimentant d'aquestes dades, tal com seria una comunicació real entre elements d'una xarxa.

Una altra implementació de la nostra simulació seria poder realitzar per part del router, el protocol DHCP. En el nostre projecte partim que tots els elements de la simulació se'ls hi assigna una IP en funció d'unes característiques ja explicades en capítols anteriors. Doncs bé, una nova funcionalitat seria que el router assignés les adreces IP a tots els nodes que estiguessin connectats a la seva xarxa. Per a realitzar aquesta tasca els routers haurien de posseir una taula de reserves de direccions IP, per saber quines adreces ja té assignades i quines són lliures per a futures connexions.

5.Apèndix

Ajuntarem un exemple de simulació amb els diferents quadres de diàleg amb la sortida de cada un dels protocols de cada nivell.

La simulació que utilitzarem són quatre nodes, dos hubs i dos routers connectats de la forma que veiem a continuació:

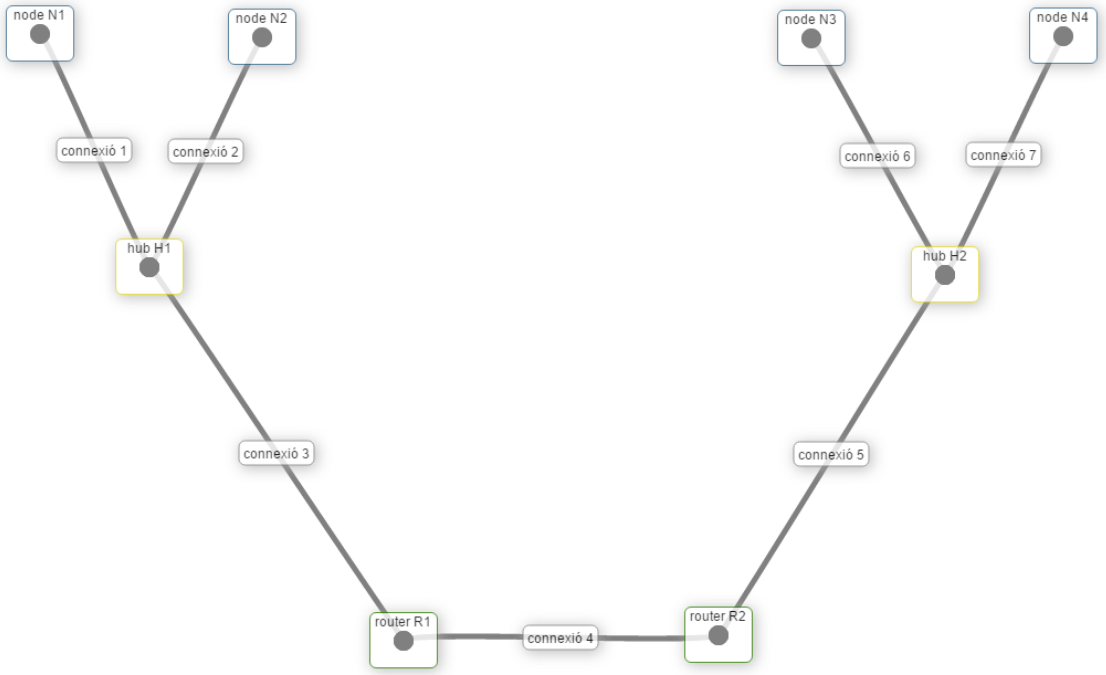

Fig. 5.1 – Simulació d'exemple – 2 nodes, 2 hubs i 2 routers

Marquem el node N1 com a emissor i envia dos missatges amb una de 5 bits cada un al node N4, que el marquem com a receptor.

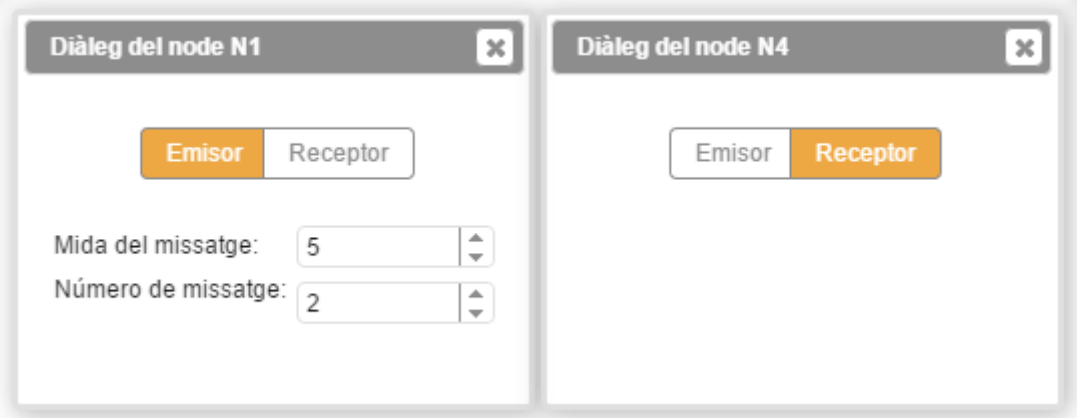

Fig. 5.2 – Simulació d'exemple – emissor i receptor

A continuació adjuntarem els quadres de diàleg resultants de la simulació que hem emprat:

Node N1

Fig. 5.3 – Simulació d'exemple – Node N1 connexió 1

Node N2

Fig. 5.4 – Simulació d'exemple – Node N2 connexió 2

Node N3

Fig. 5.5 – Simulació d'exemple – Node N3 connexió 6

Node N4

Fig. 5.6 – Simulació d'exemple – Node N4 connexió 7

Ara mostrarem varis quadres de diàleg pels hubs i els routers ja que disposen de vàries connexions respectivament. Concretament obtindrem una quadre de diàleg per a cada connexió.

Hub H1

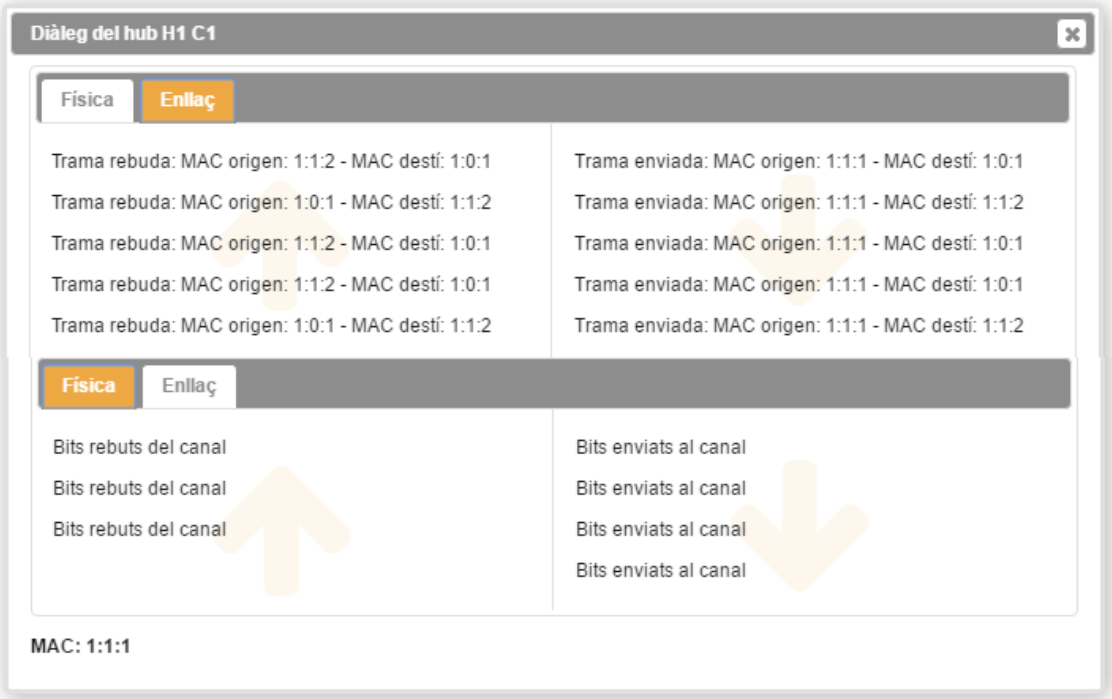

Fig. 5.7 – Simulació d'exemple – Hub H1 connexió 1

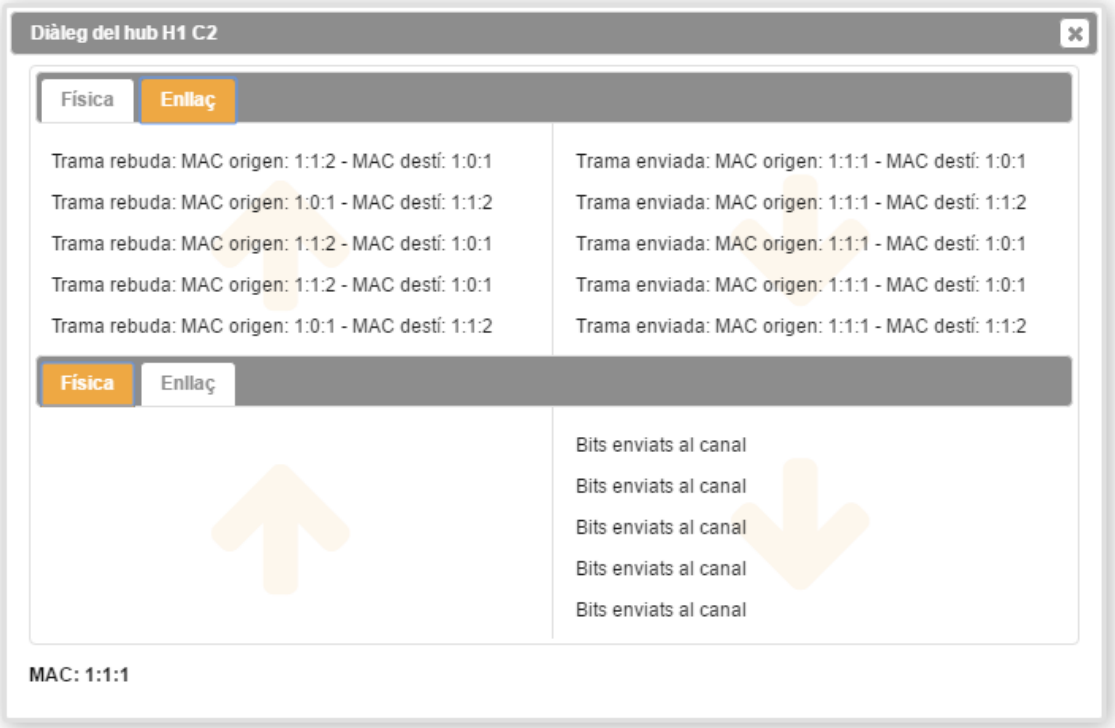

Fig. 5.8 – Simulació d'exemple – Hub H1 connexió 2

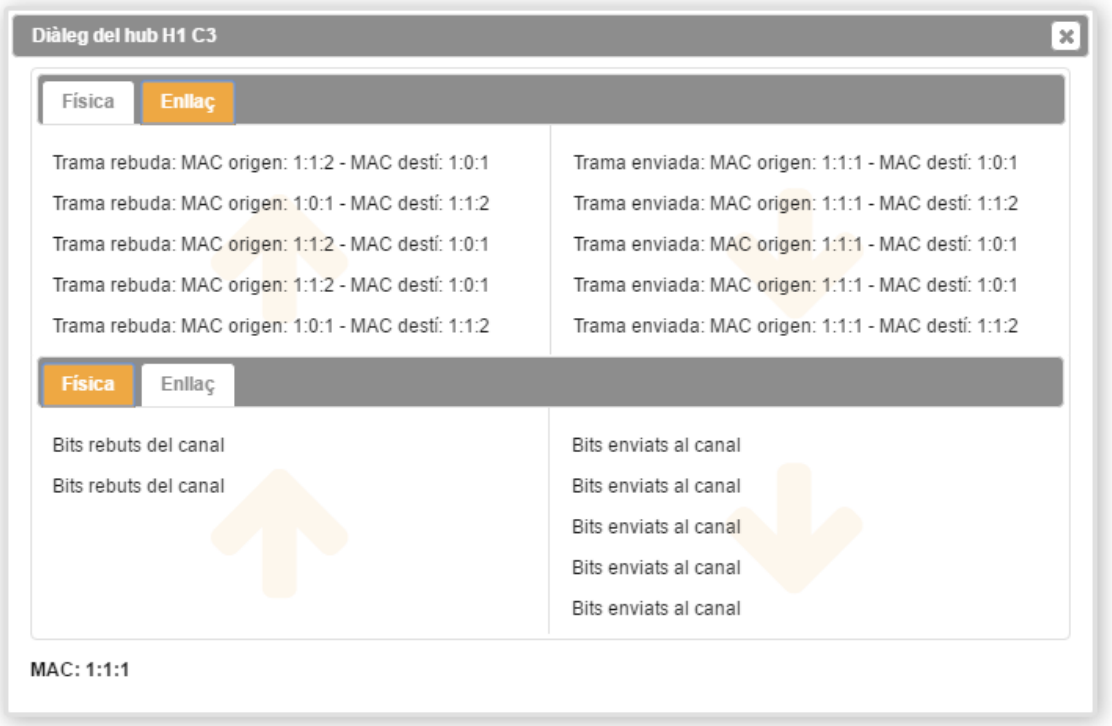

Fig. 5.9 – Simulació d'exemple – Hub H1 connexió 3

Hub H2

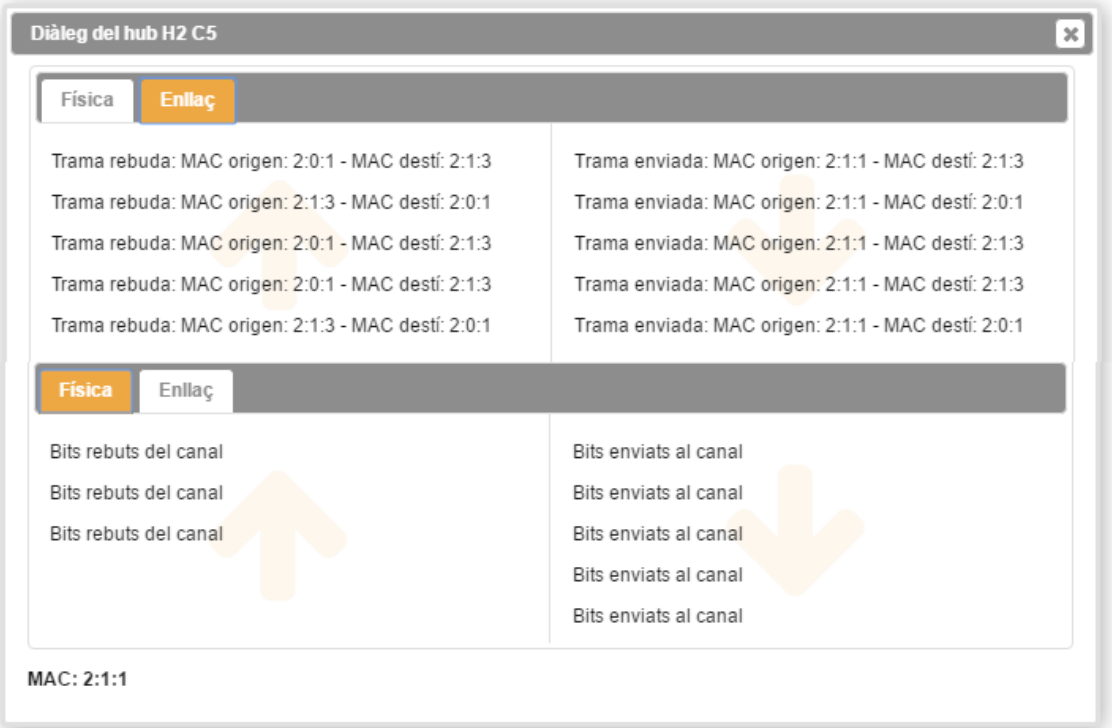

Fig. 5.10 – Simulació d'exemple – Hub H2 connexió 5

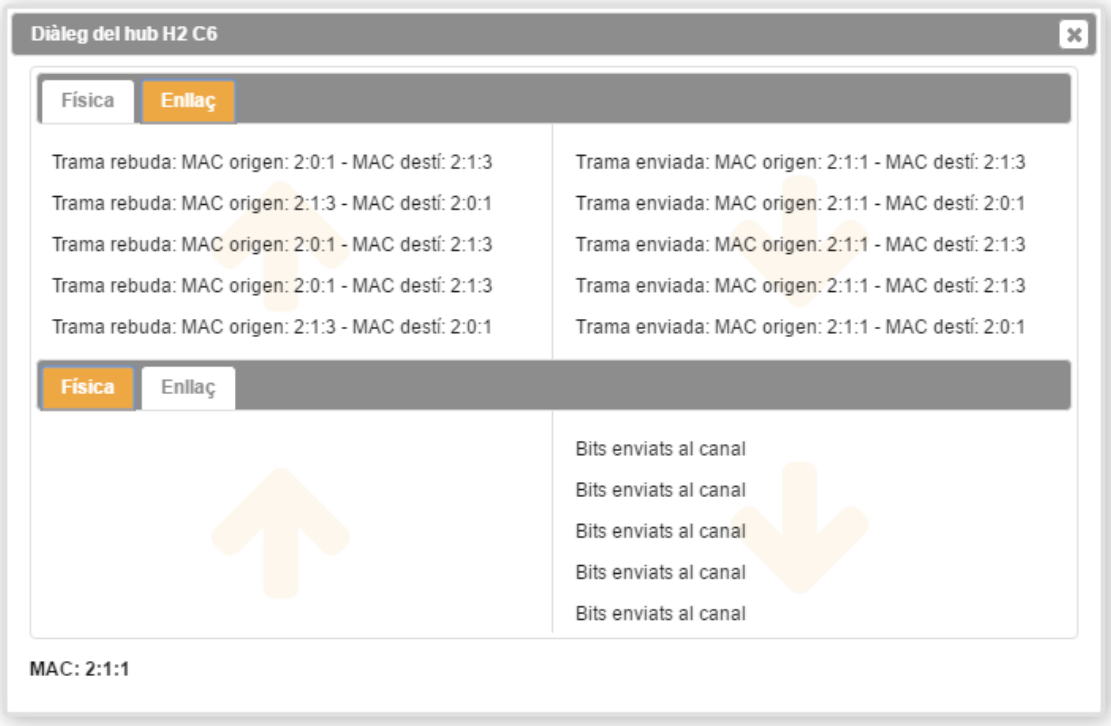

Fig. 5.11 – Simulació d'exemple – Hub H2 connexió 6

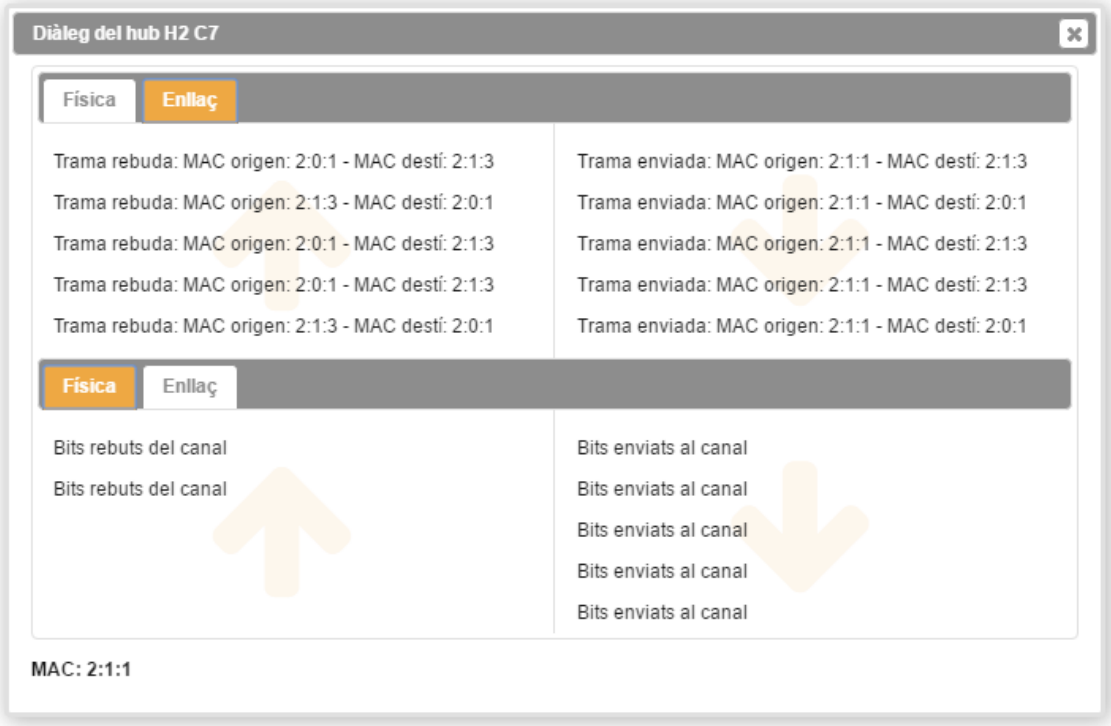

Fig. 5.12 – Simulació d'exemple – Hub H2 connexió 7

Router R1

Fig. 5.13 – Simulació d'exemple – Router R1 connexió 3

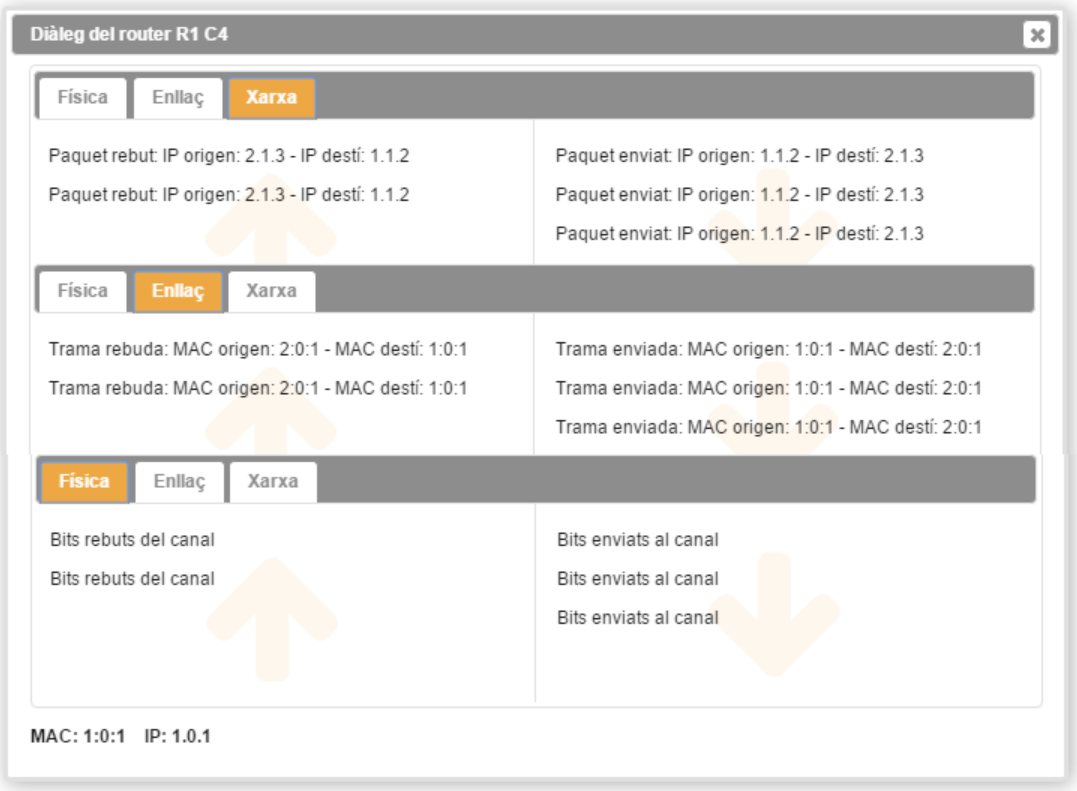

Fig. 5.14 – Simulació d'exemple – Router R1 connexió 4

Router R2

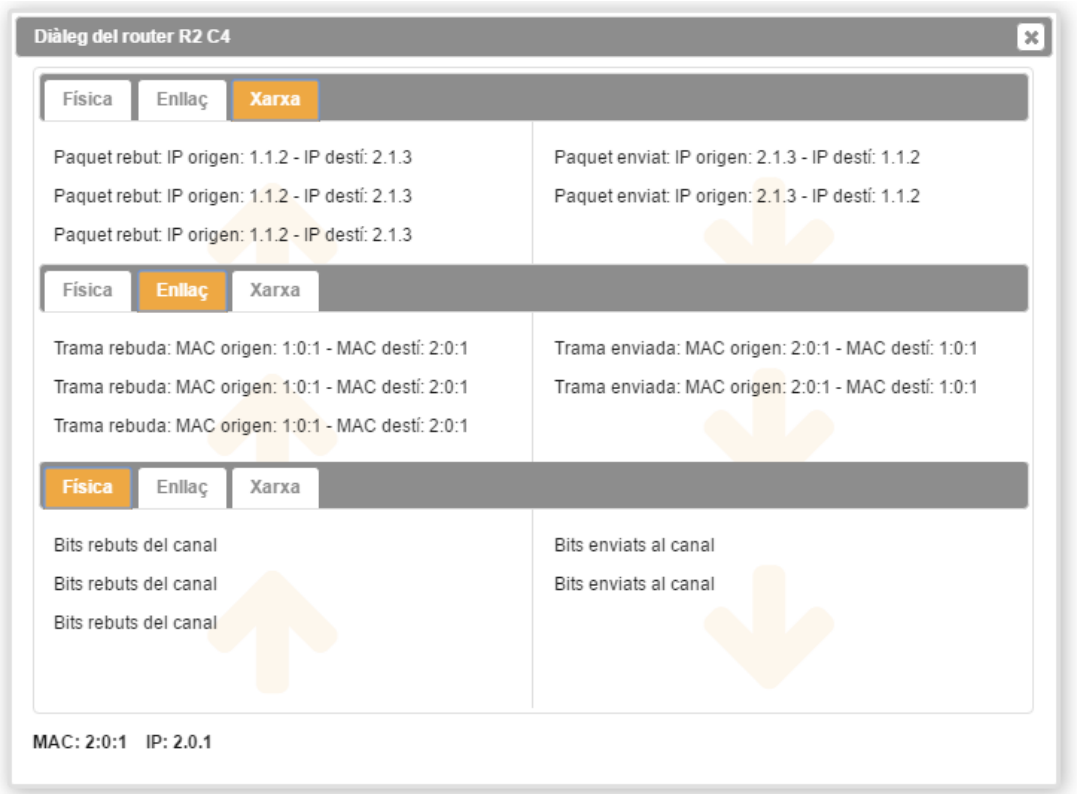

Fig. 5.15 – Simulació d'exemple – Router R2 connexió 4

Fig. 5.16 – Simulació d'exemple – Router R2 connexió 5

6.Referències

A continuació detallarem tota les referències que hem utilitzat en aquest projecte.

1. TANENBAUM, ANDREW S. (2003). Redes de Computadoras 4a edición. Ed. PRENTICE HALL.
2. APACHE. http://apache.org
3. APACHE TOMCAT. http://tomcat.apache.org
4. ECLIPSE JUNO. https://eclipse.org/juno/
5. ECLIPSE IDE FOR JAVA EE. http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/junosr2
6. JAVA EE. http://www.oracle.com/technetwork/java/javaee/overview/
7. SERVLET JAVA. http://docs.oracle.com/javaee/7/api/javax/servlet/Servlet.html
8. PROGRAMACIÓ CONCURRENT http://soft0.upc.es/pc/
9. MYSQL. https://www.mysql.com
10. JAVA API. https://docs.oracle.com/javase/7/docs/api/
11. WEBSOCKET JAVA API. https://docs.oracle.com/javaee/7/api/javax/websocket/package-summary.html
12. GSON JAVA API. https://github.com/google/gson
13. MYSQL CONNECTOR. https://www.mysql.com/products/connector/
14. JAVASERVER PAGES TECHNOLOGY. http://www.oracle.com/technetwork/java/javaee/jsp/index.html
15. HTML5. http://www.w3schools.com/html/html5_intro.asp

16. HTML WEBSOCKET API. https://developer.mozilla.org/es/docs/Web/API/WebSocket
17. HTML FILE API. https://developer.mozilla.org/es/docs/Using_files_from_web_applications
18. JAVASCRIPT http://www.w3schools.com/js/
19. JSPLUMB. https://www.jsplumbtoolkit.com
20. JQUERY. https://jquery.com
21. JQUERYUI. https://jqueryui.com
22. I18NEXT. http://i18next.com
23. JSON http://www.w3schools.com/json/
24. CSS http://www.w3schools.com/css/default.asp
25. XML http://www.w3schools.com/xml/